SecUAV - A Unified Testbed for the Evaluation of Secure State Estimators

A Master's Thesis Submitted to the Faculty of the
Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya

Author:
Oliver Ploder

In partial fulfillment of the requirements for the degree of
MASTER IN TELECOMMUNICATIONS ENGINEERING

Advisor:
Amr Alanwar[1]

Supervisor at the Universitat Politècnica de Catalunya:
Francesc Rey[2]

[1]Networked & Embedded Systems Laboratory (NESL)
University of California Los Angeles (UCLA)

[2]Signal Processing and Communications Group
Department of Signal Theory and Communications

Barcelona, July 2018

Statutory Declaration

I hereby declare that the thesis submitted is my own unaided work, that I have not used other than the sources indicated, and that all direct and indirect sources are acknowledged as references. This printed thesis is identical with the electronic version submitted.

Signature

Place, Date

Declaración jurada

Declaro bajo juramento que la tesis entregada resulta de mi propio trabajo y sin ayuda ajena. Además, que no he usado otras referencias y fuentes bibliográficas que las indicadas y que todas las fuentes directas e indirectas son reconocidas como referencias. Esta tesis impresa es idéntica a la versión electrónica presentada.

Firma

Lugar, Fecha

Abstract

Rapidly evolving technology and tight coupling of physical sensors and actuation with the system behind it in Cyber Physical Systems (CPS) have opened up new forms of security vulnerabilities to adversaries. These new types of attacks on the physical (PHY) layer can be used to drive the system in an undesired, unstable state which could ultimately resolve in life-threatening consequences for users. This new research-area has gained traction in the CPS community recently, but a feasible solution has yet to be found. One problem with the current approach of proposing algorithms and counter-measures against such attacks is, that it can be hard to compare different works in terms of effectiveness and computational cost as every research group has different ways of providing results. This work introduces SecUAV, common testbed for the CPS community. SecUAV utilizes OpenUAV, an open-source UAV simulator, for providing a common testbed for the *fair* evaluation of secure state estimation algorithms. This is achieved by expanding the already existing functionality with a framework allowing to simulate different kinds of attacks and to deploy different secure CPS algorithms. The feasibility of SecUAV is demonstrated by deploying three works in it, and comparing the resulting data in a *fair* manner. One of the studies algorithms, ReCaP [1], has been developed and evaluated by SecUAV during the work on this thesis.

Contents

1. Introduction

1.1. Status quo and the importance PHY level security

The last decades have witnessed rapid development and evolution of various technical fields, one of which are Cyber Physical Systems (CPS). CPS are characterized by a close integration of physical and software components utilizing both to operate. One example would be Wireless Sensor Networks (WSN) offering many advantages and services in industry, home automation, military operations, homeland security, emergency and rescue and habitat monitoring. It typically consists of a set of distributed nodes, equipped with sensors and communication equipment, to monitor and communicate the state of the network [2]. Another example are current, and future, vehicles, which are becoming "smarter" everyday, providing a system capable of sensing its surroundings and taking necessary control actions [3]. The current trend in this matter is to move from a typical, isolated, control system to open automotive architectures which would enable new services such as vehicle-to-vehicle communication or Over The Air (OTA) software updates to improve the reliability and quality of such systems [4,5].

Due its nature, attacks on CPS may have life threatening consequences as they are able to actuate within physical spaces. An example of the recent past, of how vulnerable such systems can be, is the Stuxnet attack on Supervisory Control and Data Acquisitions (SCADA) controllers for industrial processes [6,7]. It has further be shown [8–10], that today's vehicles can be exploited and are not resilient against attacks. An attacker therefore may be able to disrupt the system and critical components of the car. In the future, self driving cars will become widely available and more sophisticated than they are today, giving the problem even more importance [11]. Even more dangerous than compromised self driving cars, a military drone operating under the control of an adversary can be a powerful and deadly weapon [12,13]. Lastly, spoofing GPS signals to misdirect vehicles off course [14] or to take over an Unmanned Aerial Vehicle (UAV) [15] can result in damages to humans and equipment alike. Further, other types of attacks on a variety of cyber physical systems have been shown in [16–18].

To protect against such attacks, a paradigm shift in security needs to happen, as traditional – cryptographic based – mechanisms do not provide sufficient protection. These techniques cannot protect against physically compromised signals or a manipulated environment around the sensor node. To solve this problem, researchers aim to tackle the problem with secure state estimation [19] where the target is to accurately estimate the system's state out of corrupted measurements. Another approach is intrusion detection of sensors, actuators and communication network attacks.

1.2. The need for a unified CPS Security framework

With these issues in mind, researchers throughout the CPS community started and tried to find a solution to these problems with various applications in mind. Several propositions have been made to close these vulnerabilities, but the problem lies in comparing them. Different researchers usually have different approaches of showing that their solution is feasible for solving the problem at hand, CPS security (e.g. by means of MATLAB simulations, convergence analysis, implementation on a custom testbed, etc.). The issue, for industry and academics alike, arises when trying to find out if one of the many proposed approaches would work for a given system. As the process of getting results varies greatly from paper to paper, it can be hard, especially for those without a strong background in programming, to *fairly* compare the different approaches in order to find the best suitable for a given application. This is especially important, considering that the services offered by CPS require strict timing constraints and accurate estimates during operation. Further, CPS usually have power consumption constraints [20], limitations in used bandwidth [21], computation limitations [22] and generally very limited resources available for counter mechanisms. Therefore a unified framework for fairly evaluating proposals is the key to enable rapid development of efficient and reliable secure state estimators.

This thesis is therefore split into four parts:

- Revision of OpenUAV and introduction of the newly proposed SecUAV .
- Overview of three proposed solutions to the issue which are going to be *fairly* compared.
- Implementation, measurements and simulation of these three works in the proposed framework.
- Analysis of the obtained results and discussion of the process.

It has to be noted, that part of work related to this thesis was devoted in configuring and programming SecUAV as well as work on ReCaP [1] (mainly development of the algorithm and evaluation in SecUAV). This work is not explicitly mentioned throughout this document.

1.3. Thesis Organization

Chapter 2 introduces the notation and terminology used throughout this work before explaining the problem statement and a brief background of localization. The simulation environment and SecUAV are discussed and explained in **Chapter 3**, while **Chapter 4** provides the reader with a summary of the three secure state estimator proposals considered in this work. The specific case studies and results are discussed in **Chapter 5**, before providing a conclusion and outlook in **Chapter 6**.

1.4. Notation and terminology

Throughout this thesis, the transpose of a matrix and vector are denoted by $\mathbf{A}^{\mathrm{T}}$ and $\mathbf{x}^{\mathrm{T}}$, respectively, while $\mathbf{A}^{-1}$ denotes the inverse of a matrix. The $n \times n$ identity matrix is given by $\mathbf{I}_n$, a row or column vector of size n containing all ones or zeros is denoted by $\mathbf{1}_n$ and $\mathbf{0}_n$, respectively. In general, the estimation of a magnitude $\mathbf{x}$ is denoted by $\hat{\mathbf{x}}$. Further, $\mathbf{A} \bullet \mathbf{B}$ represents the element wise product of vectors or matrices. For a set $\mathcal{S}$, $|\mathcal{S}|$ denotes its cardinality and for sets $\mathcal{S}$ and $\mathcal{R}$, $\mathcal{S} \backslash \mathcal{R}$ corresponds to the elements of $\mathcal{S}$, that are not in $\mathcal{R}$. The complement of a set $\mathcal{K} \subset \mathcal{S}$ is given by $\mathcal{K}^{\mathcal{C}} = \mathcal{S} \backslash \mathcal{K}$. The i^{th} element of a vector $\mathbf{x}_k$ is either given by x_k^i or $x_{k,i}$, depending on the context. Additionally, $|\mathbf{x}|$ and $|\mathbf{A}|$ denote a vector and matrix, whose elements are the absolute values of $\mathbf{x}$ and $\mathbf{A}$, respectively. For matrices $\mathbf{P}$ and $\mathbf{Q}$, $\mathbf{P} \preceq \mathbf{Q}$ specifies that the matrix $\mathbf{P}$ is element-wise smaller or equal to $\mathbf{Q}$. Furthermore, for a vector $\mathbf{e} \in \mathbb{R}^p$, the support of the vector is given by the set

$$\operatorname{supp}(\mathbf{e}) = \{i \big| \mathbf{e}_i \neq 0\} \subseteq \{1, 2, \ldots, p\},$$

and the l_0 norm is the size of $\operatorname{supp}(\mathbf{e})$, i.e. $\|\mathbf{e}\|_{l_0} = |\operatorname{supp}(\mathbf{e})|$. Lastly, the l_0 norm of a matrix $\mathbf{E}$ is defined as $\|\mathbf{E}\|_{l_0} = |\operatorname{rowsupp}(\mathbf{E})|$, where

$$\operatorname{rowsupp}(\mathbf{E}) = \{i \big| \mathbf{E}_i' \neq 0\} \subseteq \{1, 2, \ldots, p\},$$

with $\mathbf{E}_i'$ corresponding to the rows of the matrix $\mathbf{E}$.

2. Problem statement

The specific problem statement, is being described in more detail in the following section. Afterwards, a brief introduction to localization will be given as it will serve as a basis for the chosen works explained in Chapter 4 as well as the test cases studied in Chapter 5. This chapter further aims to give the reader a better understanding on the choices made with regard to simulator and SecUAV.

2.1. Overall problem

With the rise of autonomous systems (e.g. self-driving cars, industry automation, UAVs, etc.) crucial aspects of the operation of such CPS are based on readings from various sensors (e.g. sonar, RADAR, LiDAR). This opens up a new vulnerability in these kind of systems for potential adversaries – seeking to drive the system into an undesirable state – by manipulating the readings, a firmware (or operating system) receives from the sensors. This can be achieved, for example, by physically tampering with sensors, like positioning an object statically in front of a RADAR antenna, or injecting false information directly in the sensor feed (i.e. a man-in-the-middle attack).

Generally, the firmware uses these readings to estimate a state, enabling the system to be modeled as a non-linear time varying system as

$$\begin{aligned} \mathbf{x}_{i+1} &= F(\mathbf{x}_i) + \mathbf{n}_i \\ \mathbf{y}_i &= H(\mathbf{x}_i) + \mathbf{v}_i \end{aligned}, \tag{2.1}$$

where $\mathbf{x}_i$ is the state and $\mathbf{y}_i$ is the measurements vector out of the attacked sensors at time i of size $n \in \mathbb{R}$ and $m \in \mathbb{R}$, respectively. $F : \mathbb{R}^n \to \mathbb{R}^n$ and $H : \mathbb{R}^n \to \mathbb{R}^m$ are the state- and output update function, respectively. The process and measurement noise are denoted by zero-mean Gaussian random variables $\mathbf{n}_i$ and $\mathbf{v}_i$, where $\mathbf{Q}_i$ and $\mathbf{R}_i$ are the process noise and the measurement noise covariance matrices at time i, respectively.

If an attacker chooses to tamper with the measurements $\mathbf{y}_i$ at any given time, this might yield in a (wrongfully) estimated state $\hat{\mathbf{x}}_i$ and therefore could result in catastrophic consequences for the CPS. Researchers therefore try to find means of nullifying such attacks in a way, that the actual state of the system can be recovered regardless of it being tampered with or not. This is commonly modeled by modifying (2.1) to read

$$\begin{aligned} \tilde{\mathbf{x}}_{i+1} &= F(\tilde{\mathbf{x}}_i) + \mathbf{n}_i \\ \tilde{\mathbf{y}}_i &= H(\tilde{\mathbf{x}}_i) + \mathbf{v}_i + \mathbf{a}_i \end{aligned}, \tag{2.2}$$

where $\mathbf{a}_i$ corresponds to an attack at time step i. Modeling the attack as an additive term for the measurements $\mathbf{y}_i$, enables to model a malicious node (i.e. the node directly reporting false readings) or corrupted link (e.g. via a man-in-the-middle attack).

Considering (2.1) being the uncompromised system yielding in correct state estimates $\hat{\mathbf{x}}$, the goal of a secure state estimator (i.e. secure CPS algorithm) is to obtain the true state out of the system (2.2) in case $\mathbf{a} \neq \mathbf{0}_m$. Of course it should also be able to identify that the system is not under attack if $\mathbf{a} = \mathbf{0}_m$, i.e.

$$\hat{\hat{\mathbf{x}}} = \hat{\mathbf{x}} \quad \forall\, \mathbf{a} \in \mathbb{R}^m\,, \tag{2.3}$$

as long as the sensor readings (without attack in (2.2)) are the same for both systems.

Naturally, such general systems can handle a variety of applications, for example, localization, time synchronization, control and tracking of system parameters (such as position of a part of a robot, velocity, acceleration, etc.) or advanced models. This, inevitably, leads to proposed CPS algorithms to be able to handle just as many different problems. Thus, common metrics can be hard to clarify and a *fair* comparison of different works can be a difficult task to achieve. This work therefore defines two metrics being used to obtain a *fair* comparison:

- **Estimation error**: The difference between the actual state of the system (ground truth) and the estimated state.
- **Execution time**: The time needed to estimate the system's state per time step based on the measurements received up until the current iteration.

This thesis therefore aims to offer a solution to the problem of not being able to compare different works in a *fair* and easy way, by providing a framework to achieve this task with these metrics in mind.

2.2. Localization

As discussed in Section 2.1, problems considered by different secure CPS algorithms cover a variety of forms and applications. In an attempt to not shift the focus of this work on the specific applications of the discussed algorithms, one common issue is picked, serving as the underlying problem that needs to be solved.

The chosen application serving as a basis for the *fair* comparison therefore is **localization** of an object (car, UAV, robot, sensor, etc.) in a defined region (e.g. a room, warehouse, open field). This task can be accomplished by a multitude of ways, a common approach is to obtain distance measurements to/from objects at known locations (anchors) and using these to triangulate the position of the object in the global reference frame.

It is important to note, that localization problems are usually non-linear systems with regard to (w.r.t.) the distance measurements. As there is very little work on secure CPS algorithms available – at the moment of writing this thesis – being able to deal with non-linear problems, an additional, linear formulation, based on [23], has been implemented.

Both approaches will be explained in the following after a brief introduction on how to obtain distance measurements.

2.2.1. Obtaining distance measurements

Measuring the distance to an object can be done in a multitude of ways, these mechanisms are divided into two categories, active and passive, for the remainder of this work to ease understanding.

2.2.1.1. Active measurements

Active measurements require the object to which the distance shall be determined to be involved in the process. This includes, for example, a base station, another car or UAV which either transmits information periodically (beacons) or when inquired by the mobile node which would like to know the distance. Common mechanisms in this category include:

- **Received Signal Strength Indication (RSSI)**: In this case, an anchor periodically broadcasts its own location to its surroundings, a receiving entity can use the strength of the received signal to estimate its distance to the anchor. This approach requires knowledge of the wireless channel on which the signal is transmitted for accurately estimating the range [24].
- **Time Of Arrival (TOA)**: In this case, an anchor sends a packet including a time stamp in a periodic manner, or when requested. The target then uses the included time and the time of arrival to determine the air time, and ultimately the distance, the packet has traveled. This approach suffers accuracy in case the two systems do not have a perfectly aligned clock [25].
- **Time Difference Of Arrival (TDOA)**: Like TOA, TDOA is based on the air time of packets. This approach however usually starts with the mobile node to start a chain of communication which usually includes two to four messages to be exchanged. This allows both, anchor and node, to determine their ranges from each other. Further, due to the mechanism the ranges are obtained, clock synchronization issues cause less of a concern than for TOA [25].

While the Global Positioning System (GPS) is part of this type of measurements as well, as satellites periodically send signals used for triangulation by the mobile node, it is not further studied or used in the remainder of this work. The reason for this choice is, that although its capabilities are well studied and proven, attacks on this system have been shown in [14,15]. Further, devices equipped with a GPS receiver require significantly more space and energy compared to the remaining sources of active distance measurements. Even though this does not pose a problem in systems like cars, boats or airplanes the compromises required by incorporating GPS into sensor networks or other forms of energy sensitive and space limited systems may not be possible to be made for such solutions [25].

2.2.1.2. Passive measurements

On the other hand, passive ways of determining a distance to an object only require the mobile node to conduct measurements without interaction of an anchor or other obstacles. This is useful if anchors, or obstacles, can not interact with the environment or it is not feasible to do so, examples include: walls, trees, the ground or landmarks. Using passive measurements can also be of interest if the other party is not trusted because it is known to falsify active measurements. Technologies in this category include laser, RADAR, LiDAR and sonar. All of these technologies rely on sending a signal and measuring the time it takes for it to arrive back at the mobile node's receiver in order to determine the distance to an object.

2.2.2. Nonlinear approach

Regardless of how measurements are obtained, the task of actually determining the location of a mobile node is a nonlinear problem with regard to the acquired ranges in the form of (2.1). An commonly applied approach is to use the conducted data for triangulation in which the distances serve as radii of circles (2D) or spheres (3D) with the center being the known locations of the anchors. The point in which all intersect is the position of the mobile node. This requires three or four measurements, for 2D and 3D, respectively, to obtain a unique solution [25] which can be seen in Figure 2.1 for the 2D case.

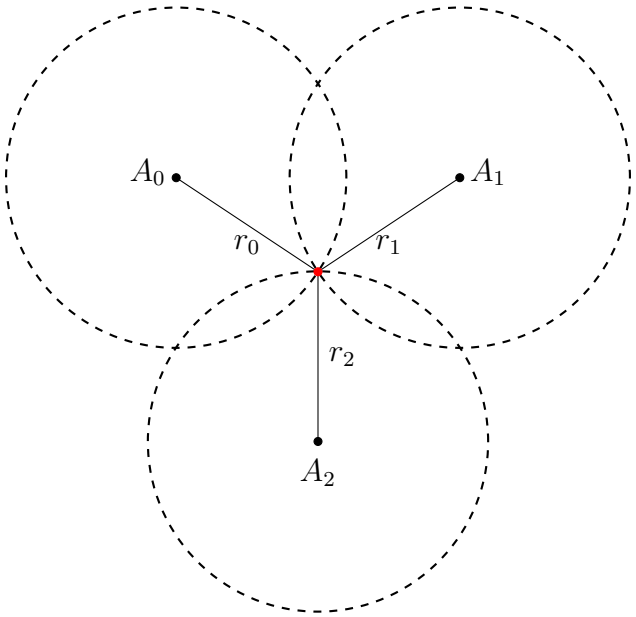
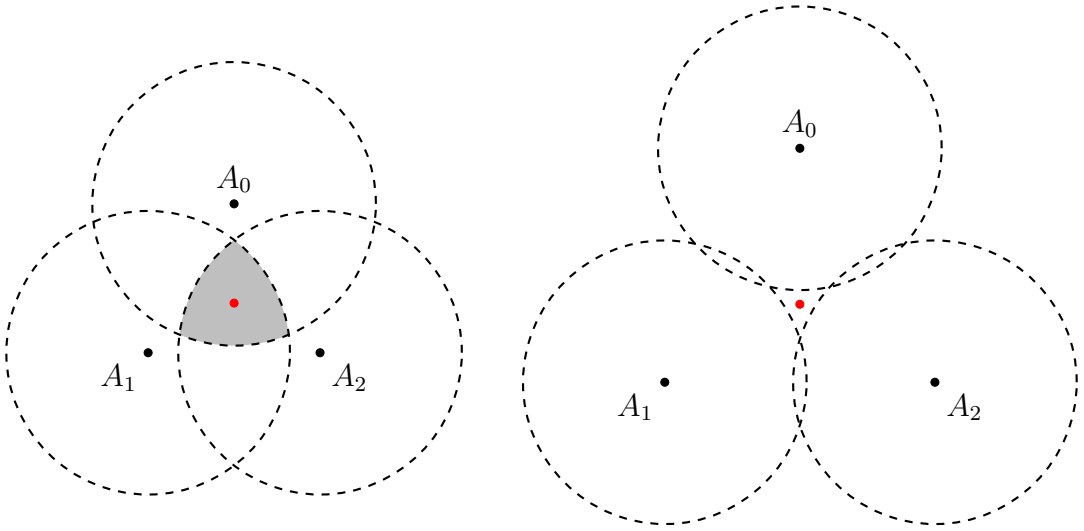


Figure 2.1.: Using distance measurements from three anchors to determine the 2D-position of a mobile node (red) by triangulation.

If the distances to the anchors are subject to noise, or if the clock sources are not perfectly synchronized (in case of TOA, TDOA) the result is, however, not a single point, but an area in which the target lies (c.f. Figure 2.2a). In this case it can further happen, that the circles (or spheres) do not intersect or at least span a general area in which the node could lie (depicted in Figure 2.2b).



(a) (b)

Figure 2.2.: Noisy readings on the distance measurements between mobile node (red) and anchors A_i can yield in (a) a general area (gray) in which the node could lie or (b) no solution at all, as the circles spanned by the readings do not intersect at the same point, nor span an overlapping area.

Using TOA for 2D localization of one mobile node determining its distance to three anchors (base stations) at known locations, the authors of [25] showed that the problem can be formulated as

$$\begin{bmatrix} x_2 & y_2 \\ x_3 & y_3 \end{bmatrix} \begin{bmatrix} x_m \\ y_m \end{bmatrix} = \frac{1}{2} \begin{bmatrix} (x_2^2 + y_2^2)^2 - r_2^2 + r_1^2 \\ (x_3^2 + y_3^2)^2 - r_3^2 + r_1^2 \end{bmatrix} , \tag{2.4}$$

where r_i is the measured distance from node m to anchor $i \in [1, 3]$, x_j and y_j are the known 2D-locations for anchors $j \in [2, 3]$ and $[x_m, y_m]^{\mathrm{T}}$ is the unknown position of the mobile node. This formulation is based on the assumption that the first anchor is located at the origin $[x_1, y_1]^{\mathrm{T}} = [0, 0]^{\mathrm{T}}$.

Rewriting (2.4) as

$$\mathbf{H}\,\mathbf{x} = \mathbf{b}\,, \tag{2.5}$$

with

$$\mathbf{H} = \begin{bmatrix} x_2 & y_2 \\ x_3 & y_3 \end{bmatrix} , \mathbf{x} = \begin{bmatrix} x_m \\ y_m \end{bmatrix} , \mathbf{b} = \frac{1}{2} \begin{bmatrix} (x_2^2 + y_2^2)^2 - r_2^2 + r_1^2 \\ (x_3^2 + y_3^2)^2 - r_3^2 + r_1^2 \end{bmatrix} . \tag{2.6}$$

allows to formulate the least-squares solution to (2.4) as

$$\hat{\mathbf{x}} = \left(\mathbf{H}^{\mathrm{T}}\,\mathbf{H}\right)^{-1}\,\mathbf{H}^{\mathrm{T}}\,\mathbf{b}\,. \tag{2.7}$$

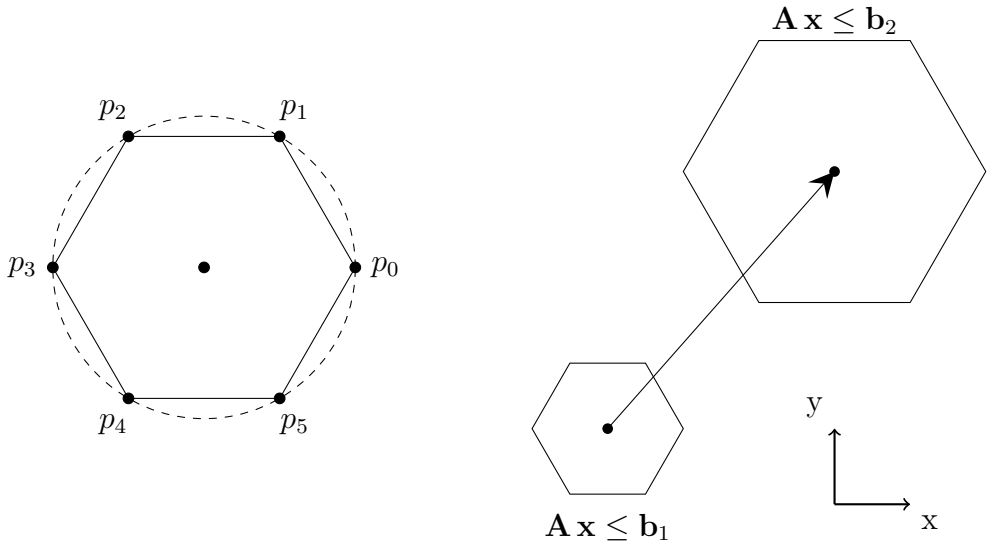
It can be clearly seen that the ranges r_i are included in the problem (2.4) and the solution (2.7) in a quadratic manner, therefore the system can not be modeled linearly but only in the nonlinear form (2.1).

2.2.3. Linear localization

Most secure CPS algorithms are currently only able to deal with linear systems in the form

$$\begin{aligned} \mathbf{x}_{i+1} &= \mathbf{A}\,\mathbf{x}_i + \mathbf{n}_i \\ \mathbf{y}_i &= \mathbf{C}\,\mathbf{x}_i + \mathbf{v}_i \end{aligned}\,, \tag{2.8}$$

with $\mathbf{A}$ and $\mathbf{C}$ being the state and output update matrices, respectively. A linearization of the problem has therefore been applied in an attempt to be able to *fairly* compare the different approaches regardless. One way of achieving this is to use polyhedra based localization, as it is described in [23]. The idea behind this approach is to use discretely sampled circles (forming a polyhedra, see Figure 2.3a) instead of the circle itself (as described before) for the localization of a mobile node.



(a) (b)

Figure 2.3.: Discretization of a circle by a 6 sided polygon (a) and polyhedron discretization with two different vectors $\mathbf{b}$ and fixed $\mathbf{A}$ (b).

A polyhedron is the intersection of n half spaces and can be represented as

$$\mathcal{P}(\mathbf{A}, \mathbf{b}) = \left\{ \mathbf{z} = (x, y) \in \mathbb{R}^2 \middle| \mathbf{A}\,\mathbf{z} \leq \mathbf{b} \right\} \qquad \text{with} \qquad \mathbf{A} = \begin{bmatrix} \mathbf{a}_1^{\mathrm{T}} \\ \vdots \\ \mathbf{a}_n^{\mathrm{T}} \end{bmatrix}, \mathbf{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}, \tag{2.9}$$

where $\mathbf{A}$ is the collection of the unit normal vectors and $\mathbf{b}$ the offsets of the polyhedron, respectively. While $\mathbf{A}$ represents the shape of the polyhedron, the location and scale are determined by the vector $\mathbf{b}$, which can be seen in Figure 2.3b.

In the case of localization, m anchors are used to determine the position of the target node. Hence, a set of m circles can be discretized into m polyhedra, constructed by $\mathbf{A}_i$ and $\mathbf{b}_i$ for $i \in [1, m]$. The intersection of the m polyhedra creates another polyhedron

$$\mathcal{P}_{\mathbb{T}} = \left\{ \mathbf{z} = (x, y) \in \mathbb{R}^2 \middle| \bar{\mathbf{A}}\,\mathbf{z} \leq \bar{\mathbf{b}} \right\} \qquad \text{with} \qquad \bar{\mathbf{A}} = \begin{bmatrix} \mathbf{A}_1 \\ \vdots \\ \mathbf{A}_m \end{bmatrix}, \bar{\mathbf{b}} = \begin{bmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_n \end{bmatrix}, \tag{2.10}$$

representing the intersection of the m other ones. The target location $\hat{\mathbf{x}} = (\hat{x}, \hat{y}) \in \mathbb{R}^2$ now lies within $\mathcal{P}_{\mathbb{T}}$, which can be obtained by solving

$$\hat{\mathbf{x}} = \underset{\hat{\mathbf{x}}}{\operatorname{argmin}} \|\bar{\mathbf{A}}\,\hat{\mathbf{z}}_{\mathbb{T}} - \bar{\mathbf{b}}\|_2^2 . \tag{2.11}$$

The least-squares solution to this problem is

$$\hat{\mathbf{x}} = \left(\bar{\mathbf{A}}^{\mathrm{T}}\,\bar{\mathbf{A}}\right)^{-1}\,\bar{\mathbf{A}}^{\mathrm{T}}\,\bar{\mathbf{b}} . \tag{2.12}$$

As the distances to the anchor nodes are now encoded, linearly, solely in $\bar{\mathbf{b}}$, the solution (2.12) can be represented as a linear system (2.8).

3. Simulator

Evaluating algorithms on actual hardware (e.g. RC-car, car, UAV, etc.) is undoubtedly a necessary task in order to ultimately prove feasibility of a proposed solution on the target architecture. Before actually moving to an implementation on it, it is advisable to first run simulations in a setting closely resembling the target architecture. However, as – especially while the implementation is still under development – bugs might result in a high bill to pay if things do not go as planned (e.g. a car crashing into another one or an UAV flying into an obstacle). To avoid such situations, a number of simulation environments have been developed in the recent past allowing to simulate robots and cars alike. Along autonomous cars, UAVs have become increasingly popular over the last decade, especially because, due to their design, they allow to fulfill tasks such as steady photography, hovering, deployment of sensors and many more, which can hardly be achieved by a traditional helicopter or fixed wing vehicles.

Due to both fields being very active research topics in academia as well as industry, environments which simulate either have been investigated for their capabilities and one has been chosen as a starting point to develop SecUAV . This chapter is organized as follows, in Section 3.1 requirements of the final environment are introduced and three different simulators are briefly presented and Section 3.2 motivates and explains the details of the finally chosen environment, lastly Section 3.3 introduces SecUAV and explains its architecture, capabilities and extensions.

3.1. Candidates

3.1.1. Requirements

Before presenting the three candidates to serve as a basis for this work, the key aspects on which the final choice has been based shall be discussed. Firstly and probably most importantly, as the aim is to extend a given implementation it is necessary for its code to be **open source**. Closed source systems would greatly increase the complexity of the proposed solution and limit flexibility. Another important point is **simplicity** of the platform – in terms of usage – so that not much previous knowledge is needed for setting up the system, preferably this just necessitates the end-user to download a package and execute a simple chain of commands to be able to start a simulation. This last point further implies a **well documented** and, ideally, **feature rich API** to ease extension as well as **cross-platform compatibility** so that the barrier of entry is further reduced. The ideal candidate should further offer access to a **variety of sensors** (e.g. RADAR, LiDAR, GPS, laser, Gyrometer, etc.) or should at the very least facilitate **easy implementation**

of new sensors. Lastly, in order to be the most **flexible** in terms of target applications, platoon driving of multiple cars or flying of more than one drone should be supported.

Simply put, a flexible, cross-platform open-source platform, which is well documented and supports a variety of target applications by offering to interface a multitude of sensors and simulations of more than one vehicle at a time would be the perfect candidate.

3.1.2. CARLA

CARLA [26] is a simulation framework built on top the Unreal Engine 4 (UE4) which takes care of computing the physics and graphics of the simulation. As the name suggests, the goal is to provide an environment to simulate autonomous cars. At the time of writing this thesis, CARLA is under heavy development, therefore, unfortunately, its capabilities are rather limited, not many sensors are available (currently only several cameras as well as LiDAR have been implemented). A further limitation is, that it is currently not possible to simulate more than one autonomous car in the same simulation, making it not possible to simulate platoon driving scenarios. Ultimately, the usage of distributed algorithms, where multiple cars work together on a problem, is not possible at the current state of the project.

3.1.3. Apollo

Apollo [27], like CARLA, is a framework built for autonomous cars, with the ultimate goal of deploying developed applications directly into a self-driving car. For this purpose it is built on top of an extended version the Robot Operating System (ROS), called RTOS. Due to its heavy industry support, multiple sensors are currently available, for both simulation and deployment. As the system is meant to be actively deployed into a car the project is naturally extensive and not necessarily easy to understand and/or adapt.

3.1.4. OpenUAV

OpenUAV [28] is built on top of ROS, alongside other software, as well. It aims to provide an easy to use platform for researchers and students to simulate UAVs. Due to its underlying software, it supports a multitude of sensors and is very flexible in regards of extensions. Further, it supports the simulation of multiple UAVs at once, enabling to study more complex scenarios like UAVs flying in formation. Additionally, like Apollo, the platform aims to be deployable to actual hardware, so that a simulated scenario, once giving satisfying results, can be directly tested on real UAVs without much modification.

3.1.5. Comparison

With the three simulation environments briefly explained and the defined set of requirements in Section 3.1.1, it is possible to choose a platform to build SecUAV on top of. The comparison is summarized in Table 3.1 and shows that all three candidates support

Table 3.1.: Comparison of the three target simulators against the requirements identified.

	CARLA	Apollo	OpenUAV
Open-Source	✓	✓	✓
Cross-Platform	✓[1]	✓	✓
Simplicity	✓	✗	✓
Documentation	✓	✓	✓
Rich API	✓	✓	✓
Number of Sensors	Few	Many	Many
Platoon Driving	✗	N/A[2]	✓
Flexibility	✓	✓	✓

most of the requirements, but both self-driving car platforms (CARLA and Apollo) are either too complex (Apollo) to allow a simple start or do not support many sensors and/or platoon driving (CARLA).

In light of these results, the decision ultimately fell on OpenUAV to be the target platform on top of which SecUAV shall be built. An additional advantage of OpenUAV in comparison to CARLA and Apollo is that it does not require a powerful back-end server to communicate with in order to run simple simulations. Therefore it allows researchers, and especially students, to run simple scenarios locally, rather than having to rely on powerful and expensive infrastructure.

3.2. OpenUAV

As indicated before, UAVs are of growing interest for industry, academics, the robots- and CPS communities due to their capabilities. Compared to helicopters, UAVs, or multi-rotors, allow precise movement and steady hovering with comparable simpler hardware and mechanics. All these capabilities come with a price though, a small mistake can easily yield a very expensive damage to the UAV, for example by flying into a building or tree. Since the hardware can be rather expensive, algorithms deployed on such devices should be bug free, since trial and error is not an option, especially when using an indoor environment.

The authors of [28] created OpenUAV in order to provide a framework that allows development of algorithms directly on the target platform without having to take the risk of destroying it. They further aim to thereby setting the ground stone for university courses dealing with UAVs, as their system enables a low barrier of entry to students and researchers alike. No powerful PC is required and if a mistake is made the results are just a learning experience, as no damage is done to expensive hardware.

The basis for OpenUAV is formed bye ROS [29] and MAVROS [30], which allows great

[1] It is worth noting, that a server/client architecture was chosen for CARLA, both of which can be used on different systems.

[2] No information has been found that suggests whether this functionality is supported.

flexibility, a big community and the support to deploy the system directly on supported hardware. The flexibility is given by the concept of nodes (one component, e.g. a sensor, corresponds to one or more nodes) which communicate via topics (a URI like concept). For the simulation itself, Gazebo [31] is used as physics and visualization engine. Lastly, a modified version of PX4 [32], a popular autopilot, is used to further simplify the control of the UAV. For example, the user may hand the flight controller a set of way points and PX4 takes care of controlling the rotors and remaining hardware of the UAV to fly to them. Additionally, Schmittle et al. designed OpenUAV in a Container as a Service (CaaS) architecture by utilizing Docker [33], therefore allowing it to run on all common operating systems. A more detailed explanation on the utilized frameworks and tools is given in Appendix A.1 and has been omitted here to enhance the readability.

3.2.1. Simulation environment

As already discussed, the developers of OpenUAV meant to create an easy to use platform with minimal barrier of entry, so that more people can conduct UAV research and courses without having to spend a fortune on specialized and fragile hardware. The remainder of this section is devoted to describing the architecture of OpenUAV and on how the components described so far play together in the big picture. Lastly, entry-points for the extension are shown and discussed. A short outlook on further capabilities of OpenUAV is given in Appendix A.2 to highlight the variety of applications that can be realized with this simulation tool; further strengthening the reasoning behind choosing it over CARLA and Apollo.

3.2.1.1. Architecture

OpenUAV is composed of three major components, the core OpenUAV server hosting the simulation, a front-end interface not necessarily hosted on the same system and the communication mechanism allowing the two previous components to communicate as well as providing the communication structure for the user to communicate with the front-end. Each of those is described in the following.

Additionally, the OpenUAV architecture allows for full Software In The Loop (SITL) simulations, where everything is being simulated right on the system that it is running on. It further supports Hardware In The Loop (HITL), for which the simulation does not run on some server or workstation, but directly on the target hardware. HITL can be especially useful in order to determine performance bounds and limitations. Due to one simulation requiring one target platform however, this goes against the scalability and cloud principles aimed for by the authors (e.g. increasing awareness of an UAV simulation environment to drive researchers and students alike to this area, not everyone has several UAVs – connected together – sitting in a data center for running simulations). Therefore, the focus in this work is on SITL, as it is one of the main objectives of OpenUAV.

OpenUAV Server Component

In order to achieve maximum flexibility in both terms of host systems as well as scenarios and hardware, OpenUAV utilizes the software stack composed of Docker, ROS (with MAVROS), Gazebo and a modified version of PX4 (OpenUAV Firmware [34]). This is done by combining all of these systems into one docker container (*openuav1* henceforth) as can be seen in Figure 3.1. By doing so, one container can be instantiated whenever a simulation is run, as one container usually corresponds to a single simulation, and destroyed when the simulation is finished.

By choosing this architecture for OpenUAV, the developers were able to provide complete isolation of the simulation testbed with the communication and front-end components by forwarding simulation data to ports of each container which can then be accessed on these ports. This CaaS architecture allows for maximum efficiency and flexibility, as resources are only occupied when they're really needed. Further this allows scaling of a cluster should there be the need for more simulations at the same time. Additionally, should an end-user decide to setup the whole framework locally, no complex installations of dependencies and tools is necessary, everything that is needed for this container is provided by a single dockerfile.

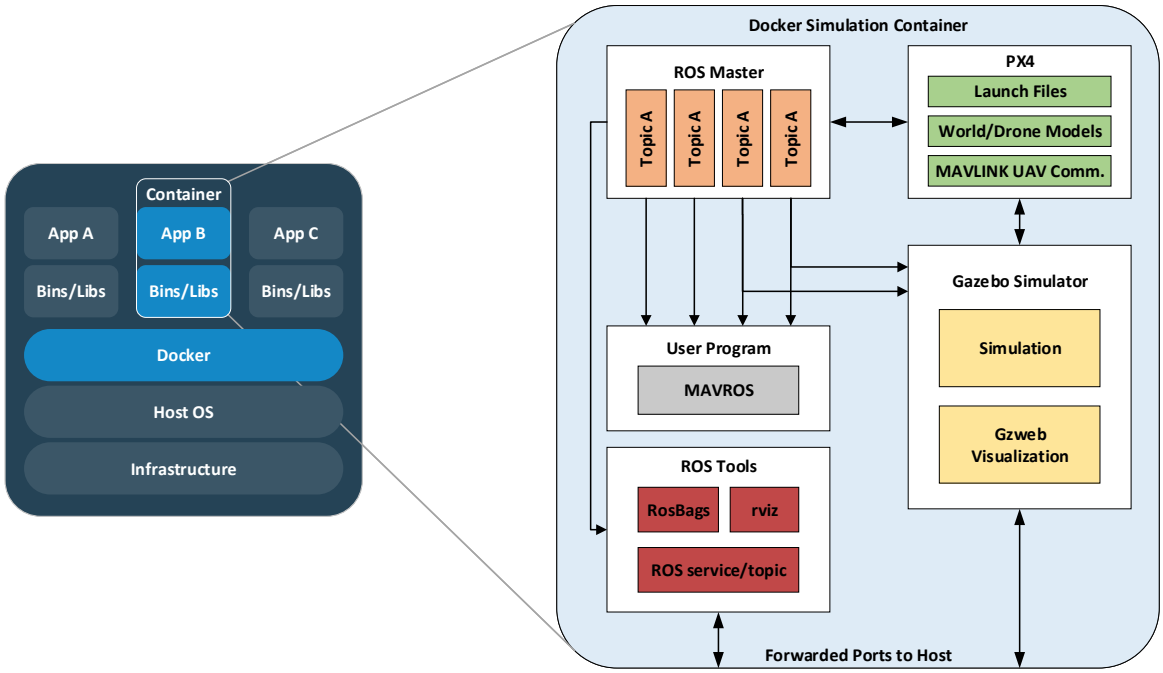


Figure 3.1.: The architecture of OpenUAV as a single docker container integrating ROS, PX4 and Gazebo [28].

Front-end Interface

The core container (*openuav1*) is supported by two other containers, one containing a database for logging and supporting the third container which is a django based dashboard so that the simulation can be observed in real time by the means of a simple browser (c.f. Figure 3.2). These two containers are responsible for user authentication, and serve as a clean interface to the simulation container(s). In addition to visualization, the front-end provides a terminal which can be used for debugging.

In case of the deployment done by the authors of [28], the front-end is hosted on the Cyber-Physical Systems Virtual Organization (CPS-VO) [35] website.

Communication Component

One of the main contributions of OpenUAV is that it is cloud enabled. This means that a user can access it anywhere, anytime, provided the system is deployed in the cloud. The communication component is responsible for handling communication between the front-end and the back-end simulation core, this is achieved via ssh, which is not a scalable form of secure access. The communication between user and front-end satisfies the requirements for cloud connectivity and is scalable. The two parts of the communication interface can be seen in Figure 3.2, the user has to authenticate itself at the front-end interface while the front-end itself authenticates on its own to the back-end, both these mechanisms make use of SSL.

During the simulation, the back-end continuously sends information to the front-end in order to display real-time video and sensor data to the user. Post-simulation ROSbags (i.e. logs from ROS) are sent to the user by *openuav1* for further evaluation once a simulation is finished.

Summary

If deployed like Schmittle et al. meant it to be, the end user (i.e. student or researcher) does not need to install any software on their machine, all they need to do is transfer their scenario to a server hosting the OpenUAV platform. Once transferred, they send a request to the web server to start a simulation, which in turn scales up an *openuav1* container and the progress can be viewed through the web dashboard. This scheme can be seen in Figure 3.2.

All of this enables a very low barrier of entry, as a scenario can be configured by means of a simple shell script. If more than the already provided functionalities are needed for a scenario, the user has the option to write their own ROS node in either C++ or Python, therefore providing maximum flexibility. Furthermore, the CaaS architecture ensures that the system is very adjustable in terms of deployment patterns, for example the system can be deployed on a high performance server (i.e. "the cloud"), scaling simulation (i.e. core) containers as they are needed for simulations, or a user can choose to run the whole

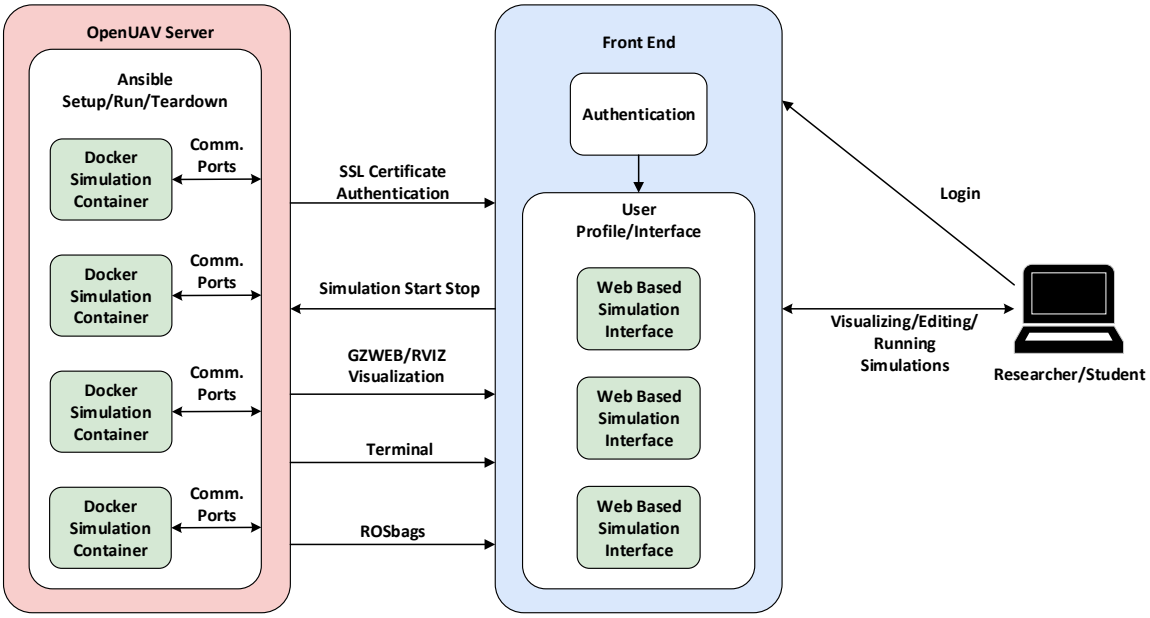


Figure 3.2.: The three components, master container, front-end and communication interface, working together on a simulation [28]. One server component represents one simulation, while the other two parts of the system do not need to be scaled. The simulation can be observed remotely via a browser interface.

platform on their own workstation, if not too much computing power is needed for a simulation.

3.2.1.2. Extendability

Having introduced the architecture of OpenUAV, the main question was, which ways there are for integrating SecUAV into it. Due to the software stack OpenUAV is based on, several points of entry have been identified, each having their own set of advantages and disadvantages. The identified ways to extend OpenUAV are discussed and compared in this section, before a final choice for SecUAV is made. The main requirements for it are the ease of use and flexibility in terms of (programming) knowledge required by the user and adaptiveness to different solutions w.r.t. different simulations, respectively.

Firmware

Possibly the first way of extending OpenUAV any developer would think of, is to modify the Firmware itself. The biggest advantage with this way is, that changes can be made at a very low level and can therefore be optimized the most so that they fit exactly what has to be achieved. While this may sound very appealing, the disadvantages outweigh the advantages by far. Because the firmware itself needs to be modified, each time a change is made, the container needs to be re-created, which takes significantly more time than any other approach, this further limits flexibility, as each feature needs to be implemented on

this low level, the container needs to be re-created and further testing needs to be done to be sure the firmware as a whole still works as expected. Because the work is done on the firmware itself, the developer doing so needs to be very proficient in C++ and needs to have an excellent understanding of a huge project, which limits this approach to only excellent programmers, rather than people with limited experience in C++.

Gazebo and ROS

Another way to implement new changes on a slightly higher level than the modification of the OpenUAV firmware, would be to change, adapt or add implementations of sensors and actuators in ROS or Gazebo. This, again, would need a re-creation of the whole container together with testing, verification and debugging of the changes. If the change is done in Gazebo, this would also imply that the implementation has only been made for the simulation environment, as Gazebo does not run on the target hardware, should that be the aim of a user. If ROS is adapted, it would work on the hardware as well as the simulation, depending on the capabilities of the hardware of course. Hence, each change in Gazebo would also ultimately require the same change in ROS as well. The main drawback – again – is that the user would need to be an expert programmer in order to create these changes in feasible time.

ROS nodes

Due to the usage of MAVROS and PX4, it is possible to simply write a new node for the software. This means that none of the underlying frameworks needs to be touched, implying that the container does not need to be recreated as inclusion of custom nodes is achieved during the startup phase of a simulation. This approach further allows the user to operate on a high level in either Python or C++, further reducing the proficiency needed to adapt a simulation. Therefore, making it possible to create new nodes that implement a custom firmware, sensors, actuators and/or a (secure) CPS algorithm without having to know the details of the underlying firmware. Therefore, writing a new node for each change required, allows rapid development and testing as well as a very adaptive environment. If one functionality is not needed for a simulation the node implementing it can simply be removed or decoupled from that simulation, and re-used in another. No bloat is given to the software as a whole as only required things stay in the final implementation, hence keeping it simple.

A simplified visualization of this concept is depicted in Figure 3.3, where all main components of the simulation are being taken care of by the master node. The master node corresponds to the default functionality OpenUAV provides out of the box. Test cases and additional functionality are defined and implemented in user nodes which communicate with the master node at a given update rate.

Due to these reasons, it was decided to extend OpenUAV by providing a set of ROS nodes that offer the functionality needed in a simple way. The disadvantage of having less control in terms of fine tuning are exceeded by the advantages and comply the most with

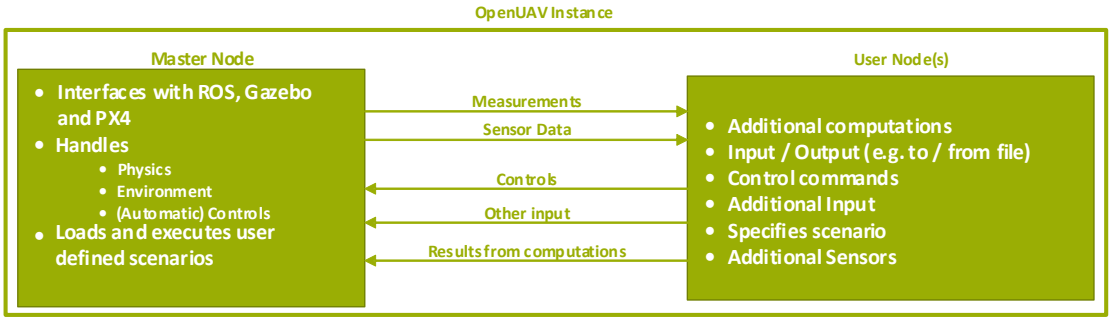


Figure 3.3.: The split between user and master space in one OpenUAV simulation instance. The master node takes care of all main responsibilities of the simulation, while user node(s) add additional functionality to the simulation.

our goal of providing an easy to use platform, enabling as many users as possible to use it without having to read and study for days and weeks just to write a simple application.

3.3. SecUAV

Having discussed OpenUAV, its capabilities and possibilities offered by its modular design, the main contribution of this thesis, SecUAV shall be introduced and its architecture, features and goal will be explained. The design of this contribution aims to keep the original goals of OpenUAV, as explained in Section 3.2, while also aiming to provide:

- A common framework for the CPS community to build, test and *fairly* compare their solutions.
- A single point at which the user can define attacks and (secure) CPS algorithms.
- Pre-defined attacks on several sensor types.
- Sample implementations of secure CPS algorithms (see Chapter 4).

Ensuring that these points are still valid in the final implementation of the framework, a toolbox for industry and research shall be provided, to compare and test different proposals in a *fair* way. The idea of including pre-defined attacks in the package is twofold, for once we would like to start a discussion in the community on how an attacker on different systems can be accurately modeled (as there is currently no common model on how to achieve this), and secondly, these pre-defined attacks – together with the sample implementations of the algorithms in Chapter 4 – shall serve as an initial starting point to users new to the platform, ensuring a rapid and steep learning curve in how to use and expand the framework.

With these goals in mind, an architecture has been designed to offer the target functionalities while still being flexible enough to enable modifications and extensions in the future. Therefore, the framework consists of three main modules all playing together when acti-

vated. The configuration of these is done by a single configuration file. Namely, the three modules are (i) a module acting as an adversary, pre-defined attacks can be enabled and the user is given the option of implementing a custom attack type if the pre-defined ones do not suffice for a scenario, (ii) a secure CPS algorithm module, which is the place where users can deploy their secure algorithms for tests and evaluation and (iii) a configuration module.

Before describing the modules included in SecUAV in detail, an overview of the whole system and an explanation on how it interacts with the remainder of OpenUAV will be given.

3.3.1. Overview

Considering the simplified view on OpenUAV's architecture shown in Figure 3.3, it was decided to implement SecUAV in between user- and master nodes, intercepting part of the communication between both parts. It is important to note, that SecUAV is not a separate part in this architecture, but part of the user nodes, as its configuration and implementation is, ultimately, depended on the simulation being carried out. This can be seen in Figure 3.4.

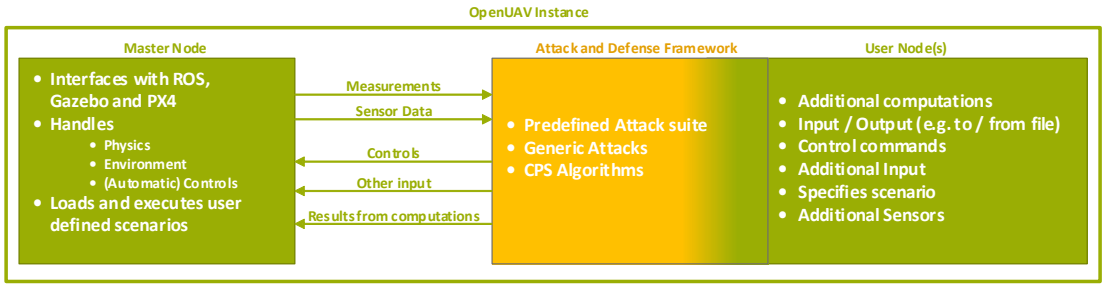


Figure 3.4.: SecUAV works as part of the user nodes, by intercepting the communication between user and master nodes, attacking and securing the communication by means of the attack framework and the deployed CPS algorithm.

Here, the attack and defense framework corresponds to SecUAV, as it is depicted in Figure 3.4, the communication can be attacked in the forward (i.e. communication from master to user node) as well as in the backwards (i.e. user node to master) paths, this ensures maximum flexibility for types of attacks and counter measures. For example, an adversary maybe attacks a sensor reading (forward path) or commands to an actuator (backwards path), both these injections could potentially threaten the state of the UAV and hence need to be considered by the framework.

3.3.2. Modules

The detailed forward path of SecUAV is shown in Figure 3.5, the user is able to configure attacks (pre-defined and custom) and choose the secure CPS algorithm to be deployed.

In order to ease the creation of a scenario before actually testing an implementation to secure the communication, it is further possible to completely circumvent SecUAV by manipulating the corresponding entries in the configuration. This shall ease the development of test scenarios. Further, it is possible to choose between the pre-defined attacks or, if these are not what the user needs, to define a custom attack type along with the CPS algorithm.

The same configuration is possible in the reverse path – although mirrored – and completely independent on the forward path. Additionally, the user can choose to deploy different attacks or none at all on various links. For example, an adversary maybe found a way to intercept a RADAR sensor, but can not fake measurements from a LASER range finder. This again, shall allow maximum flexibility in terms of tested scenarios.

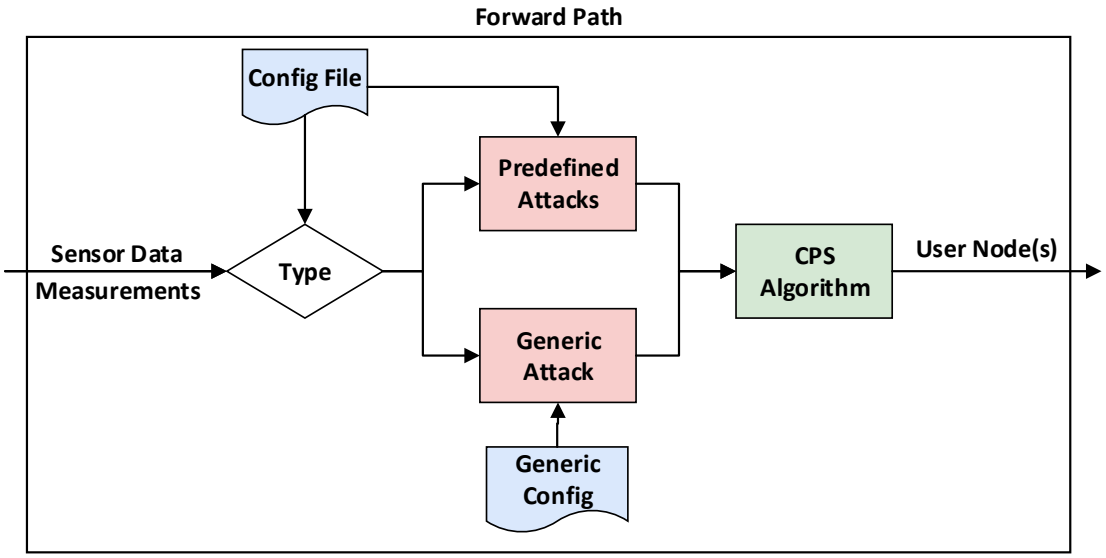


Figure 3.5.: The forward path of SecUAV , attacks and security mechanisms can be configured to act as the user needs them to. It is further possible to fully circumvent either of these so that a direct, uninterrupted, communication between user and master nodes is possible.

The communication between the different modules of the master node, framework and user node(s) works by using ROS topics, therefore, for intercepting a message, the framework subscribes to the topic which shall be attacked and publishes the attacked (and ultimately secured) signal on another topic, which can then be used by either user or master node(s) (depending on the direction of the attack).

Since the CPS algorithm module is nothing else than a ROS node providing the secure algorithm to the framework, its discussion is omitted here, as the algorithms provided are discussed in detail in Chapter 4. The remaining functionality is similar for the attack module (publishing, subscribing, etc.). Thus, the remainder of this section discusses predefined and custom attacks and finishes with an explanation of the configuration of the framework.

3.3.3. Attacks

It has already been mentioned that the attack module is mainly composed of two submodules, pre-defined and custom attacks. We were aiming to provide a generic set of attack types in order to keep the usability for a rich set of sensors high. Before describing the attack framework in detail, an attempt is been made to distinct two types of attackers.

3.3.3.1. Attack classification

In general, attackers in the presented framework are utilizing man-in-the-middle attacks to corrupt sensor readings in an attempt to – ultimately – drive the UAVs in an undesired state, i.e. bring them off their path or crash them. Attacks of this type, can be classified into two main categories, type A and type B.

Type A attack

This type of attackers aim to stay undetected by the (secured) CPS, their approach is to only inject sensor reading that would make sense with the system itself. For example, if the corrupted sensor is a velocimeter of an UAV, injecting a reading of $100\,\mathrm{m\,s^{-1}}$ if the previous (uncorrupted) measurement was $1\,\mathrm{m\,s^{-1}}$ would be easily detectable as an attack, since a UAV would not be able to fly at such speeds nor accelerate as much in a split-second. Instead, adversaries of this kind stick to the physical limitations of the system and environment in an attempt to remain hidden from the system.

This can only be achieved if the attacker knows these physical limitations. Since OpenUAV is, also, based on an open-hardware concept, availability of such information is a realistic assumption. Therefore, such attacks would not raise suspicion in the CPS as they seem just like uncorrupted signals from the system's point of view.

Type B attack

On the contrary to type A attacks, type B adversaries do not care about being detected. Therefore, they can inject whatever signals they seem fit to achieve their end-goal of maximizing damage done to the system. An attacker of this kind, would inject a sensor reading of $100\,\mathrm{m\,s^{-1}}$, even if it does not fit within the physical model and limitations of the UAV.

In general, type B attacks are harder to defend against, as there are no constraints on injected values. Furthermore, an attacker of this kind does not need to know the specifications of the system (e.g. datasheets of the used sensors), but – if known – this would make this type of assault even more dangerous to the system. For example, knowing that the insecure system directly couples measurements from the velocimeter with actuation of the rotors, makes it an easy task to destruct the system faster than a type A attacker.

3.3.3.2. Pre-defined attacks

Pre-defined attacks are limited to 6 types, split in two parts in an attempt to provide enough functionality to cover a vast set of possible applications. While the pseudocode for this module is described in Algorithm 1, the attacks of the first part are:

- A constant and customizable offset that is being added to sensor readings or actuator commands.
- A ramp with customizable slope.
- A seesaw with variable amplitude and frequency/slope.

Either one of these, or none, can be added to the second part of the attacks, which are:

- A Uniform distribution with customizable mean and variance.
- A Gaussian distribution with variable mean and variance.
- A Pareto distribution, characterized by scale and shape parameters, as an example of a heavy-tailed distribution.

The parameters of the distributions and signals in the first part can be chosen to represent a logical attack with acceptable sensor readings/actuator commands for a given scenario. It is further possible to choose a time-varying attack, i.e. changing types or parameters over time, to evaluate algorithms more throughout and to model an attacker that maybe is able to see the effect of the injected signals on the UAV, hence trying a different approach in an attempt to fool the system.

3.3.3.3. Custom attacks

If a user wishes to deploy a custom implementation of an attack, a new ROS node needs to be supplied to the framework. To ease the implementation of such a node, the existing pre-defined attack node is written as generic as possible and comments have been provided for further clarity. Once this custom implementation has been written, the name, and possible parameters need to be introduced to a custom configuration script, which can be based on the provided configuration for the pre-defined attacks.

3.3.4. Configuration

By now, a configuration script has been mentioned several times, which shall now be explained in detail. In an attempt to keep the complexity minimal for the user, a simple JSON file is used to configure the framework, as can be seen in Algorithm 2.

The sample shows the configuration options for adding time-varying attacks on links. The fields have the following meaning:

- **links**: This is the overall array of link objects. One entry in the array corresponds to one link in the simulation and is composed of all other fields explained below.

Algorithm 1 : Pre-defined attack algorithm, attacks can be chosen from two categories or none at all

procedure ADDATTACK(x, t, type_1, type_2, A, f, μ, σ)
  $x_{\text{tmp}} = x$
  switch type_1 **do**
    case 1:
      $x_{\text{tmp}} = x_{\text{tmp}} + A$
    case 2:
      $x_{\text{tmp}} = x_{\text{tmp}} + t \times f$
    case 3:
      $x_{\text{tmp}} = x_{\text{tmp}} + \text{seesaw}\,(t, A, f)$
    case default:
      // do nothing
  switch type_2 **do**
    case 1:
      $x_{\text{tmp}} = x_{\text{tmp}} + \text{uniform}\,(\mu\,, \sigma)$
    case 2:
      $x_{\text{tmp}} = x_{\text{tmp}} + \text{gauss}\,(\mu\,, \sigma)$
    case 3:
      $x_{\text{tmp}} = x_{\text{tmp}} + \text{pareto}\,(\mu\,, \sigma)$
    case default:
      // do nothing
  return x_{tmp}
end procedure

- **topicIn**: Considering the case of the forward-path being under attack (c.f. Figure 3.5), this is the name of the topic the sensor under attack publishes its data to.
- **topicOut**: The topic on which the attacked signal shall be published on. This is also the topic to which the user node(s) or CPS algorithm has to subscribe to, to get the attacked values.
- **msgType**: The type (i.e. format) of the messages published by the sensor. The same message format will be used to publish the attacked signal.
- **attacks**: The attacks being deployed on the specified link are configured in this array.
- **tFrom**: This field specifies the starting time step from which on the specific entry shall take effect.
- **tTo**: The duration of the attack on the link is defined in this field. All time durations correspond to time-steps, i.e. calls of the attack procedure in Algorithm 1.
- **type1, type2**: Specify the type of attack being used in the defined time frame. If no attack is wished, these fields need to be set to $i \notin [1, 3]$.

- **A, f, mu, sigma**: Lastly, the configuration of the various attack types is specified by these parameters, their effect is as shown in Algorithm 1.

Algorithm 2 : Sample JSON configuration for adding a time-varying attack on one link

```
{
    "links" : [
        {
            "topicIn" : "/uav1/meas/laser1",
            "topicOut" : "/uav1/meas/attack/laser1",
            "msgType" : "sensor_msgs/Range",
            "attacks" : [
                {
                    "tFrom" : 0,
                    "tTo" : 60,
                    "type1" : 1,
                    "A" : 2.5,
                    "f" : 0,
                    "type2" : 2,
                    "mu" : 0,
                    "sigma" : 1
                },
                {
                    "tFrom" : 60,
                    "tTo" : 120,
                    "type1" : 2,
                    "A" : 0,
                    "f" : 0.5,
                    "type2" : 1,
                    "mu" : 0,
                    "sigma" : 1
                },
                {
                    "tFrom" : 120,
                    "tTo" : 180,
                    "type1" : 4,
                    "A" : 0,
                    "f" : 0,
                    "type2" : 4,
                    "mu" : 0,
                    "sigma" : 0
                }
            ]
        }
    ]
}
```

If the user chooses to use a custom attack on any link, it can simply be skipped in the configuration (i.e. not include the affected links in the configuration file). Potential configurations for custom attacks can be based on the framework, enabling to keep the architecture simple and complexity low.

In the specific case shown in Algorithm 2, the link of laser1 is attacked ("/uav1/meas/laser") and the attacked signal is published on "/uav1/meas/attack/laser1", while the message is of type "sensor_msgs/Range". For the first 60 time steps, a Gaussian distribution with

zero mean and variance of 1 will be applied to which a constant term of 2.5 is added. The next 60 time steps apply a ramp with slope 0.5 and a uniform distribution with zero mean and a variance of 1. The attacker chooses to not apply an attack for the last 60 time steps, as both types are set to 4 (hence outside the range $[1, 3]$).

4. Secure state estimation algorithms

In an attempt to show the capabilities and limitations of the proposed CPS framework, three secure state estimation algorithms are deployed on the different use cases. The results will be used to *fairly* compare the three algorithms in Chapter 5.

As the test cases are based on a localization problem, papers are targeted which are able to handle nonlinear problem formulations. However, as there is currently a very limited selection of such papers, the third work considered in this thesis is based on a linear system, for which the linear approach, described in Chapter 2, will be used.

4.1. Secure State Estimation with Application to Localization and Time Synchronization (SecSens)

SecSens [36] is a collection of two secure CPS algorithms, SecEKF and SecOPT. Either of these two algorithms can be used and deployed separately, depending on the needs of the secure state estimator. SecSens can deal with type A and type B attacks, and is based on an Extended Kalman Filter (EKF) [37] (SecEKF) as well as an optimization problem (SecOPT). While SecSens can be used for different applications, the focus of the authors was to provide a framework for localization and time synchronization in a hostile environment. Although the algorithm(s) will be explained as the Alanwar et al. intended, they are only used for localization based problems, as time synchronization is not an applicable problem in OpenUAV[1]. SecOPT and SecEKF have been implemented for this thesis and shall be explained in the following subsections after providing a general overview of the framework as a whole.

4.1.1. Problem statement

SecSens considers a secure state estimation problem over a network of N nodes $k \in \{0, \ldots, N-1\}$ spatially distributed in space. Further, two nodes are considered connected if they can communicate with each other directly. The neighborhood of a node k is denoted by $\mathcal{N}_k$. This nonlinear, time-varying system under attack is modeled as

[1] In order to set up a time synchronization test case, different nodes of the scenario would need to have separate clock sources, which, due to the nature of ROS, is not the case for OpenUAV. It should be noted that it would be possible to implement such a test case in OpenUAV though, although requiring additional work (such as the extended ROS version Apollo is using) which is out of the scope of this thesis.

$$\begin{aligned} \tilde{\mathbf{x}}_{i+1}^{k} &= \tilde{f}\left(\tilde{\mathbf{x}}_{i}^{k}\right) + \tilde{\mathbf{n}}_{i}^{k} \\ \mathbf{y}_{i}^{k,j} &= h^{j}\left(\tilde{\mathbf{x}}_{i}^{k}, \tilde{\mathbf{x}}_{i}^{j}\right) + \tilde{\mathbf{v}}_{i}^{k} + \mathbf{a}_{i}^{k,j}\,, \end{aligned} \tag{4.1}$$

where $\tilde{\mathbf{x}}_{i+1}^{k}$ is the state of node k and $\mathbf{y}_{i}^{k,j}$ is the measurement sent to node k from node $j \in \mathcal{N}_k$ at time i. $\tilde{f} : \mathbb{R}^n \to \mathbb{R}^n$ and $h^j : \mathbb{R}^n \to \mathbb{R}^m$ are the state- and output update functions, respectively. The process and measurement noise are denoted by zero-mean Gaussian random variables $\tilde{\mathbf{n}}_{i}^{k}$ and $\tilde{\mathbf{v}}_{i}^{k}$, where $\mathbf{Q}_i$ and $\mathbf{R}_i$ are the process noise and the measurement noise covariance matrices at time i, respectively. The attack vector $\mathbf{a}_{i}^{k,j}$ is a vector that models how an attacker changes the sensor measurements at time i between node j and k. This enables modeling of a malicious node k and a corrupt link (k, j). Non-zero elements in the vector $\mathbf{a}_{i}^{k,j}$ correspond to the attacked values on the corresponding sensor; otherwise the measurement is not attacked.

4.1.2. Core idea

Instead of identifying links under attack and excluding them from the estimation, which is a highly complex combinatorial problem, SecSens reduces the number of observed nodes (and hence possible attacks) at time i to the set of active nodes communicating with each other. This drastically reduces the complexity and allows SecSens to operate in real time as the number of unknowns equals the number of nodes. With this idea in mind, the model (4.1) is extended with $\mathbf{x}_{i}^{k} = \left[\tilde{\mathbf{x}}_{i}^{k^{\mathrm{T}}}\ \mathbf{a}_{i}^{k^{\mathrm{T}}}\right]^{\mathrm{T}}$, yielding

$$\begin{aligned} \mathbf{x}_{i+1}^{k} &= f\left(\mathbf{x}_{i}^{k}\right) + \mathbf{n}_{i}^{k} \\ \mathbf{y}_{i}^{k,j} &= h^{j}\left(\mathbf{x}_{i}^{k}, \mathbf{x}_{i}^{j}\right) + \mathbf{v}_{i}^{k}\,. \end{aligned} \tag{4.2}$$

It is important to note, that $\mathbf{a}_{i+1}^{k} = \mathbf{a}_{i}^{k}$ are considered constant in this model, time varying aspects of an attack are considered to be included in $\mathbf{n}_{i}^{k}$. Further, SecSens differs between:

- **Outliers**: Readings which lie outside the expected target range (e.g. a reading of 100 m in a 5 m × 5 m box) are detected and rejected (i.e. not considered) for the estimation.
- **Smart attacks**: If readings are well inside the boundaries set by the system, they cannot be removed by an outlier detector. A smart attacker would therefore inject values within these limits.

This concept allows further simplification of the problem, as outliers do not have to be considered in the secure state estimation algorithms.

SecSens assumes to be able to measure time difference and distance between moving nodes in conjunction with measurement noise[2]. The measurement vector at node k, from node

[2] As mentioned in the introduction of this section, SecSens considers simultaneous time synchronization and localization as a prime example of its capabilities. It is however explicitly mentioned, that it is able to handle other applications, or a subset of the presented ones. Therefore being a good candidate for the work conducted in this thesis.

$j \in \mathcal{N}_k$ is composed as

$$\mathbf{y}_i^{k,j} = \begin{bmatrix} d_i^{k,j} + ao_i^k \\ r_i^{k,j} + ad_i^k \\ R_i^{k,j} + ad_i^k \end{bmatrix}, \tag{4.3}$$

where $d_i^{k,j}$ denotes the counter difference at time i, which is the measured difference between the clocks of each node. Further, $r_i^{k,j}$ represents the frequency bias discrepancies, modeled by a single-sided two-way range between nodes k and j. Lastly, $R_i^{k,j}$ is a second distance measurement between the two nodes based on three exchanged messages – therefore more accurate then $r_i^{k,j}$, formally called double-sided two-way range – at time i. The attacks on these three measurements are modeled in ao_i^k and ad_i^k on the counter difference and distance, respectively.

The state of node k at time i is

$$\mathbf{x}_i^k = \begin{bmatrix} \mathbf{p}_i^k \\ o_i^k b_i^k \\ ao_i^k \\ ad_i^k \end{bmatrix}, \tag{4.4}$$

where $\mathbf{p}_i^{k^{\mathrm{T}}}$ is the three dimensional position of node k at time i, o_i^k denotes the clock offset and b_i^k the clock frequency bias; both with respect to the global reference time clock.

The full network state is considered a concatenation of the states of all nodes, i.e.

$$\mathbf{x}_i = \left[\mathbf{x}_i^{0^{\mathrm{T}}}, \ldots, \mathbf{x}_i^{N-1^{\mathrm{T}}} \right]^{\mathrm{T}}. \tag{4.5}$$

Both, SecEKF and SecOPT make use of these definitions of $\mathbf{y}$ and $\mathbf{x}$, as will be seen in the remainder of this section.

4.1.3. SecEKF

The first algorithm in this collection, SecEKF, relies on the use of an EKF to estimate the true network state and corresponding attacks. It is assumed that the clock parameters evolve according to the first-order affine approximation of the following dynamics $o_{i+1}^k = o_i^k + b_i^k \, \delta_t$ and $b_{i+1}^k = b_i^k$, with $\delta_t \coloneqq t_{\mathrm{M}}(i+1) - t_{\mathrm{M}}(i)$ where t_{M} is the root node time (i.e. global time). Further, the process update function f is defined as

$$f\left(\mathbf{x}_i^k\right) = \begin{bmatrix} \mathbf{p}_i^k \\ o_i^k + b_i^k \, \delta_t \\ b_i^k \\ ao_i^k \\ ad_i^k \end{bmatrix}. \tag{4.6}$$

The measurement update function h^j for any given node k with respect to a second node j is given by

$$h^j\left(\mathbf{x}_i^k, \mathbf{x}_i^j\right) = \begin{bmatrix} \left(o_i^j - o_i^k\right) + ao_i^k \\ \left(1 + b_i^k\right) \|\mathbf{p}_i^j - \mathbf{p}_i^k\|_2 + ad_i^k \\ \left(1 + b_i^k\right) \|\mathbf{p}_i^j - \mathbf{p}_i^k\|_2 + ad_i^k \end{bmatrix} . \tag{4.7}$$

Based on these definitions, an EKF has been designed, seeking to minimize the mean-square error $\mathrm{E}\|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2$, where $\hat{\mathbf{x}}_i$ is the current estimate of $\mathbf{x}_i$. The algorithm itself follows three steps:

1. **Reception of measurement**: The estimation server has received a new measurement after time step i which is a result of a message exchange between nodes k and j, which terminates at j. This exchange may be subject to an attack, or one of the nodes is compromised.
2. **Calculation of δ_t**: The server maintains an estimate of the master time $t_\mathrm{M}(i)$ as well as the time offsets and frequency biases of all nodes. Therefore, it is able to calculate the time elapsed in between measurements in the master time frame.
3. **Time update**: The current estimates of time and location for each node are updated while taking possible attacks under consideration.

4.1.4. SecOPT

The second algorithm in the collection is based on a Maximum Likelihood Estimator (MLE), which is formulated as

$$\underset{\hat{\mathbf{x}}_i^k, a_i^k \forall k,j}{\operatorname{argmin}} \sum_{\substack{j,k \\ i' \in [i-L,i]}} \left(\|h^j\left(\mathbf{x}_{i'}^k, \mathbf{x}_{i'}^j\right) - \mathbf{y}_{i'}^{k,j}\|^2 - \lambda \|a_{i'}^k\|_1 \right) , \tag{4.8}$$

where the observed time window is of length L, i.e. L measurements are considered in each estimation step. In order to achieve a more robust attack estimation, the L1-norm is taken of the attack values. The complete algorithm is summarized in Algorithm 3 and is supposed to be deployed to a centralized location (e.g. server), like SecEKF. For $i = 0$ the state is initialized with $\mathbf{x}_\mathrm{init} = \mathbf{0}_n$ whereas all subsequent runs take the previous state estimate as an initial point.

Algorithm 3: SecOPT

Set $i = 0$ and $\hat{\mathbf{x}}_0 = \mathbf{x}_{\text{init}}$

Step 1: Collect enough measurements to fill the window.

Step 2: Solve the centralized optimization problems

$$\left[\hat{o}_{i+1}^j \, \hat{\mathbf{p}}_{i+1}^j \, \hat{ao}_{i+1}^j\right] = \underset{o_i^j \, \mathbf{p}_i^j \, ao_i^j}{\operatorname{argmin}} \sum_{j \in \mathcal{N}_k} \left(d_i^{k,j} - \left(o_i^k - o_i^j\right) - \|\mathbf{p}_i^k - \mathbf{p}_i^j\|_2/c - ao_i^j\right)^2 + \lambda \left\|ao_i^j\right\|_1 ,$$

$$\left[\hat{\mathbf{p}}_{i+1}^j \, \hat{ad}_{i+1}^j\right] = \underset{b_i^j \, \mathbf{p}_i^j \, ad_i^j}{\operatorname{argmin}} \sum_{j \in \mathcal{N}_k} \left(R_i^{k,j} - \|\mathbf{p}_i^k - \mathbf{p}_i^j\|_2 - ad_i^j\right)^2 + \lambda \left\|ad_i^j\right\|_1 .$$

Step 3: Increment i by one, go to Step 1.

4.2. Resilient Cyber Physical Systems Against Adversarial Sensor Attacks (ReCaP)

ReCaP [1][3] is another secure CPS algorithm that utilizes addition of deterministic noise to the measurements, it is making use of [38,39] and can deal with both type A and type B attacks. Unlike other algorithms introduced in this chapter, ReCaP aims to provide attack-free sensor measurements as a service to whatever industrial firmware is deployed on the CPS. In other words, an additional layer is added to the system, completely decoupling security and functionality (c.f. Figure 4.1). This approach enables quick implementation into a system without altering anything in the firmware, it further allows to keep the secure state estimation simple as will be shown later.

4.2.1. Problem statement

Similar to SecSens (in (4.2)), the system considered here is modeled as

$$\begin{aligned} \mathbf{x}_{i+1_{\text{sys}}} &= F_{\text{sys}}(\mathbf{x}_{i_{\text{sys}}}) + \mathbf{n}_{i_{\text{sys}}} \\ \mathbf{y}_{i_{\text{sys}}} &= H_{\text{sys}}(\mathbf{x}_{i_{\text{sys}}}) + \mathbf{v}_{i_{\text{sys}}} + \mathbf{a}_{i_{\text{sys}}} \end{aligned} , \tag{4.9}$$

where $\mathbf{x}_{i_{\text{sys}}}$ is the state and $\mathbf{y}_{i_{\text{sys}}}$ is the measurements vector out of the attacked sensors at time i. $F_{\text{sys}} : \mathbb{R}^n \to \mathbb{R}^n$ and $H_{\text{sys}} : \mathbb{R}^n \to \mathbb{R}^m$ are the state- and output update function, respectively. The process and measurement noise are denoted by zero-mean Gaussian random variables $\mathbf{n}_{i_{\text{sys}}}$ and $\mathbf{v}_{i_{\text{sys}}}$, where $\mathbf{Q}_{i_{\text{sys}}}$ and $\mathbf{R}_{i_{\text{sys}}}$ are the process noise and the measurement noise covariance matrices at time i, respectively. The attack vector $\mathbf{a}_{i_{\text{sys}}}$ is a vector that models how an attacker changes the sensor measurements at time i. Non-zero elements in the vector $\mathbf{a}_{i_{\text{sys}}}$ correspond to the attacked values on the corresponding sensor; otherwise the measurement is not attacked.

[3]Note that the development and evaluation of ReCaP was part of the work conducted during the course of this thesis.

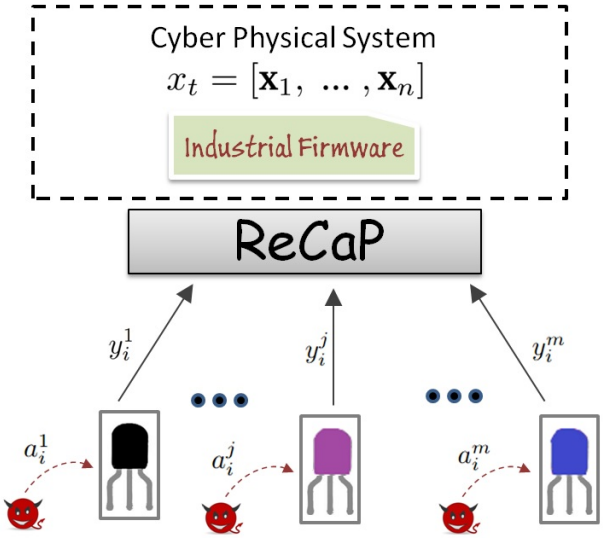


Figure 4.1.: ReCaP equips CPS with a protection layer that aims to protect industrial firmwares from sensor reading attacks. Industrial firmwares are designed to deal only with noisy sensor measurements, not attacked ones. In the depicted case, all sensors are under a time-varying attack a_i. The corrupted measurements y_i are fed to ReCaP. Then, ReCaP provides attack-free sensor readings to the industrial firmware at each time step i.

4.2.2. Core idea

The core idea is, that ReCaP makes use of the fact that some sensors can produce correlated measurements, for example a distance measurement is correlated with both, velocity and acceleration measurements. Thus, p sensors of the CPS yielding correlated signals are grouped in m sensor groups $S_{G1}, S_{G2}, ..S_{Gm}$. Additionally, a sensor group may only consist of a single sensor ($p = 1$) and can be part of multiple sensor groups (e.g. a velocity sensor can be part of one sensor group of distance sensors and another of velocity measurements). Collecting multiple sensors into a group. Note that p is not constant over all sensor groups, i.e. different groups may be composed of a different number of sensors. The following new state spaces (as seen in Figure 4.2) are proposed:

1. **State Space per Sensor Group** j: The secure state estimators' state space may be different for each sensor group. It defines the relationship between the sensor measurements in the group and takes attacks into account. For example, different types of distance sensors (laser, LiDAR, RADAR) have different noise floors. Further, it may be necessary to apply a prior transformation to some measurements – as shown in [40] – in order to yield the desired signal. For example, a camera may be used to get a distance measurement as well, but in this case the camera feed needs to be transformed before yielding in a distance measurements. This would be the state space for the gray box in Figure 4.2

$$\begin{aligned} \tilde{\mathbf{x}}_{i+1_{\mathrm{G}^j}} &= \tilde{\mathbf{F}}_{\mathrm{G}^j} \ \tilde{\mathbf{x}}_{i_{\mathrm{G}^j}} + \tilde{\mathbf{n}}_{i_{\mathrm{G}^j}} \\ \mathbf{y}_{i_{\mathrm{G}^j}} &= \tilde{\mathbf{H}}_{\mathrm{G}^j} \ \tilde{\mathbf{x}}_{i_{\mathrm{G}^j}} + \tilde{\mathbf{v}}_{i_{\mathrm{G}^j}} + \mathbf{a}_{i_{\mathrm{G}^j}} \end{aligned} . \tag{4.10}$$

2. **State Space for CPS**: The state space of the CPS (i.e. the firmware) does not

consider attacks anymore, as the measurements are considered secured by (4.10) already when reaching it. It further holds all nonlinearities, usually modeled by a mature state estimator dealing only with bounded noise and no attacks – e.g. an extended Kalman filter. ReCaP does not require modification to the original state estimators in the current industrial firmware which is the green box shown in Figure 4.2

$$\begin{aligned} \mathbf{x}_{i+1_{\text{se}}} &= F_{\text{se}}\left(\mathbf{x}_{i_{\text{se}}}\right) + \mathbf{n}_{i_{\text{se}}} \\ \mathbf{y}_{i_{\text{se}}} &= H_{\text{se}}\left(\mathbf{x}_{i_{\text{se}}}\right) + \mathbf{v}_{i_{\text{se}}} \end{aligned}. \tag{4.11}$$

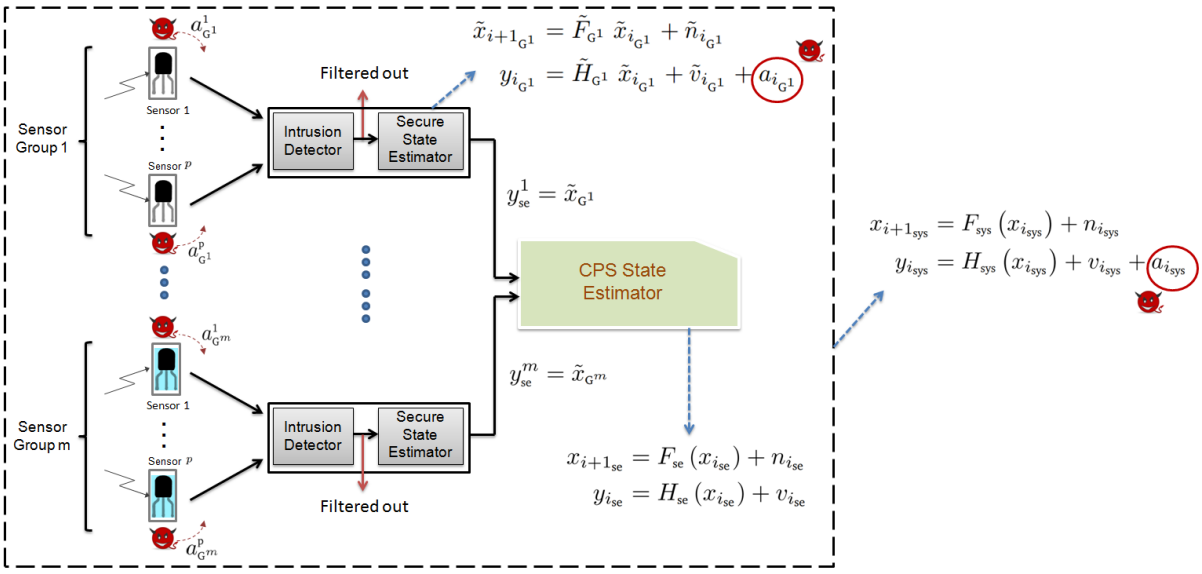


Figure 4.2.: Sensors are categorized into groups of sensors which are reporting correlated physical signals. LiDAR, sonar, camera and RADAR sensors can be used to measure the distance between cars which are complex CPS. The green box corresponds to the green box in Figure 4.1 and the same for the gray ones.

The state spaces (4.9), (4.10) and (4.11) are related by

- The secure measurements from all sensor groups are concatenated to build the secure state $\mathbf{y}_{i_{\text{se}}} = \left[\tilde{\mathbf{x}}_{i_{\text{G}^1}}, \ldots, \tilde{\mathbf{x}}_{i_{\text{G}^m}}\right]^{\text{T}}$ of the CPS state estimator in the industrial firmware.
- The system state in (4.9) is equal to the one in (4.11) and the attacked measurements in (4.10) are the same as the overall attacked measurements in (4.9), i.e. $\mathbf{x}_{i_{\text{sys}}} = \mathbf{x}_{i_{\text{se}}}$ and $\mathbf{y}_{i_{\text{sys}}} = \left[\mathbf{y}_{i_{\text{G}^1}}, \ldots, \mathbf{y}_{i_{\text{G}^m}}\right]^{\text{T}}$, respectively.

The exact algorithm is going to be described in the following, in order to ease the understanding it will be described for the one sensor group case, i.e. $m = 1$, however $p \geq 1$. Therefore the subscript G^j will be removed where there is no confusion, yielding in the simplified state space model for one sensor group

$$\begin{aligned} \tilde{\mathbf{x}}_{i+1} &= \tilde{\mathbf{F}}\,\tilde{\mathbf{x}}_i + \tilde{\mathbf{n}}_i \\ \mathbf{y}_i &= \mathbf{H}\,\mathbf{x}_i + \tilde{\mathbf{v}}_i + \mathbf{a}_i \end{aligned}, \tag{4.12}$$

with the state per sensor group $\tilde{\mathbf{x}}_i \in \mathbb{R}^n$ at time step i, with process noise $\mathbf{n}_i$ and state dynamics $\tilde{\mathbf{F}} \in \mathbb{R}^{n \times n}$. In each time step, p sensor readings $\mathbf{y}_i \in \mathbb{R}^p$ are taken. A sensor attack is modeled as an additive attack vector $\mathbf{a}_i \in \mathbb{R}^p$ along with a sensor noise signal $\tilde{\mathbf{v}}_i$.

4.2.3. Algorithm

After having discussed the basic idea of ReCaP, the framework itself is introduced in the following. Before explaining the core concepts, a new state-space model is introduced and attacks are classified. Further, in an attempt to make the notation clear in an easily grasped way, it is explained using a test case where a UAV is trying to locate itself in a room equipped with 4 anchor nodes, each time step a distance measurement to each anchor is made and its location is triangulated (see Figure 4.3).

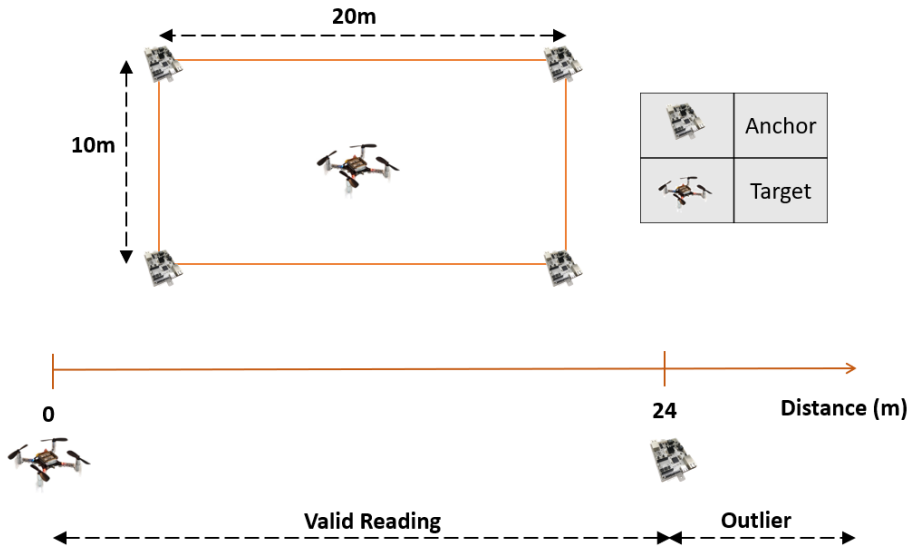


Figure 4.3.: The ReCaP framework uses the environmental prior knowledge of CPS to initially classify the compromised sensor measurements. Classifying the sensor measurement as outlier or smart-attack value.

4.2.3.1. Attack classification and modeling

Before finally discussing the main algorithm, attacks are classified and the final state-space model before the secure state estimator is introduced.

Attack classification

Sensor readings in ReCaP can be classified as one of three categories:

1. **Noisy readings**: No real system results in perfectly accurate readings, no matter how good the sensor is, a measurement will always be subject to noise. In the example shown in Figure 4.3, this type of readings correspond to the distance between the UAV and its surroundings (e.g. a wall). The ultra-wideband (UWB) transceiver [41] used in the simulations throughout this work for example, can be used to obtain a distance with an accuracy of less than 0.5 m, which means that this sensors' noise has a standard deviation of less than 0.5 m.
2. **Outlier readings**: Usually, it is possible to define clear boundaries in which a sensor reading has to lie. For example Figure 4.3 shows a relaxed boundary (to allow for some measurement noise) of 24 m, if a sensor reports a measurement that is above this boundary, it is considered an outlier (i.e. not plausible).
3. **Smart attack readings**: Readings under this category are under attack, however they are well within the boundaries set by the environment and sensor datasheets and are therefore plausible to the system.

Considering this, an attacker of type A would try to not inject outlier readings in order to aid his aim of not being detected. Attacker B on the other hand does not hesitate to produce readings that are well outside of the defined boundaries.

ReCaP can protect against both types of attacks according to its capabilities. Since outlier readings are easy to detect and therefore require different treatment than smart attacks, ReCaP is designed in a two stage way, one dealing with outliers and one with smart attack values. The authors emphasize that outlier detection is not the main problem in their work and therefore, exact boundaries are not necessary. This rather functions as a first stage before heading for the main algorithm.

Attack modeling

In order to detect smart attacks and secure the system state against them, the secure state estimator's state space consists of the CPS state as well as an estimate for the attack values(s). Alanwar et al. therefore use an augmented state vector in the form $\mathbf{x}_i = [\tilde{\mathbf{x}}_i^{\mathrm{T}}, \mathbf{a}_i^{\mathrm{T}}]^{\mathrm{T}}$, giving the modified state-space model

$$\begin{aligned} \mathbf{x}_{i+1} &= \mathbf{F}\,\mathbf{x}_i + \mathbf{n}_i \\ \mathbf{y}_i &= \mathbf{H}\,\mathbf{x}_i + \mathbf{v}_i \end{aligned} . \tag{4.13}$$

Furthermore, in order to deal with time-varying attacks, $\mathbf{a}_{i+1} = \mathbf{a}_i + \mathbf{n}_i$, factoring the variance in the attack in the model noise term $\mathbf{n}_i$ (as was confirmed in [42,43]). Lastly, ReCaP has no restriction on the statistics of $\mathbf{n}_i$ and $\mathbf{v}_i$.

4.2.3.2. The two-layered algorithm

To include both types of attackers and to yield in a robust secure state estimator, the authors consider an attacker capable of producing outlier and smart attack values alike.

The adversary further is able to attack $k = p$ sensors of a sensor group (i.e. all sensors) with k being the number of attacked sensors and p the total number of sensors. Therefore, at any given time step, k_o outlier and k_a smartly attacked readings are reported, with $k = k_\mathrm{o} + k_\mathrm{a}$.

This allows a classification of sensors at each time step i into a subset of sensors p_o reporting outliers and a subset of sensors p_a which are not reporting outliers. This means that, since $p = p_\mathrm{o} + p_\mathrm{a}$, it is possible that not all p_a sensors are under attack, but only subject to noise, the same may be said for outlier readings. ReCaP deals with both these subsets in two layers, which are going to be explained in the remainder of this section.

Intrusion Detector

The first layer is composed of any light-weight intrusion detection scheme, utilizing knowledge available to the public (e.g. datasheets, characteristics of the environment the CPS is deployed in, etc.). The main goal is to enforce the bounds (y_{τ_j} henceforth) discussed in Section 4.2.3.1 and shown in Figure 4.3.

Algorithm 4: Pseudo-code of the used outliers detector

```
Collect p measurements from the p sensors to construct y_i.
while i < END do
    temp = [ ]
    for j = 1 : p do
        if |y_i^j| < y_{τ_j} then
            temp = [temp ; y_i^j]
        end if
    end for
    y_i = temp
    p_a = size(temp)
    if p_a > 1 then
        Send y_i to Algorithm 6. The size of the parameters is adapted according to p_a.
    else
        Send y_i to Algorithm 7.
    end if
end while
```

The intrusion detector used in ReCaP, which can be replaced by any other real time state of the art detector, is summarized in Algorithm 4 and is a simple threshold detector. If the measurement y_i^j of sensor j at time i lies outside the bounds defined before (i.e. if $|y_i^j| > y_{\tau_j}$), then it is filtered out and not used anymore for the current estimation step in the second layer.

Therefore, if no attacker or only attacks of type A are present at time step i, then y_i has full size p, contrary, if attacks of type B are present or a sensor reports outlier readings, the size of y_i is $p_\mathrm{a} \leq p$. It has to be emphasized, that the resulting y_i does not need to reflect the bound extremely accurate, ReCaP can handle a loose interpretation as well.

Secure State Estimator

The second stage of ReCaP is optimization based over a sliding window $\mathcal{N}_i$ at time step i and of size w, which is composed of the measurements from a sensor group that have passed the first stage.

The algorithm makes use of several mechanisms and transformations in order to provide secure estimates to the CPS, these are highlighted in the remainder of this section.

The first idea is the use of an **adaptive initial point**, an attacker may changes their strategy over time, therefore considering previous estimate $\hat{\mathbf{x}}_{i-1}$ as the initial point in the current time step is not always the best way to go and may delays convergence. Choosing an arbitrary initial point (e.g. zero) each time step, however, may be counterproductive as well, as the history of the estimates may carries vital information. As described by Algorithm 5, a reasonable $\mathbf{x}_0$ is picked by comparing difference of the standard deviation $\sigma_{\mathcal{N}_i}$ of the current window $\mathcal{N}_i$ with the standard deviation of the previous window $\mathcal{N}_{i-1}$ to a threshold for changing strategies, σ_{τ_1}.

Algorithm 5 : Get $\mathbf{\Omega}$ and initial $\mathbf{x}_0$ at the beginning, i.e., $i = 0$.

Step 1: initialize the $\hat{\mathbf{x}} = \mathbf{0}_p$.
Step 2: Start by preparing a temporary vector **tmp** of size p_a with some perturbation weights.
for $j = 1$ **to** p_a, **stepsize** 2 **do**
  $\text{tmp}^j = \text{tmp}^j + j\ \delta$
  $\text{tmp}^{j+1} = \text{tmp}^{j+1} - j\ \delta$
end for
Step 3: Initialize the rotating weights $\mathbf{\Omega}$ which is a matrix of size $w \times p_a$ by rotating the temporary vector along the window of measurements.
for $u = 1$ **to** w, **stepsize** 1 **do**
  $\mathbf{\Omega}^{:,u} = \mathbf{tmp}$
  tmp= [tmp^{p_a} ; tmp^1 ; ... ; tmp^{p_a-1}] // rotate **tmp**
end for

The main concept of ReCaP is the use of **rotating multiplicative weights** which are used to corrupt the measurements received from Algorithm 4. The weights $\mathbf{\Omega}$ act as consistent perturbations, letting the optimizer choose not to follow the attacked readings. While constant noise would only increase the secure estimation error, consistent perturbations help the optimizer to aim for a better solution as it trusts these readings less. This is a method borrowed from machine learning [44], where it is widely used. This results in the following convex optimization problem for securely estimating the state out of the sensor readings (c.f. Algorithm 6)

$$\underset{\mathbf{x}_i}{\operatorname{argmin}} \sum_{j \in \mathcal{N}_i} \| \mathbf{\Omega}^{:,j} \bullet \mathbf{y}_j - \mathbf{H}\,\mathbf{x}_i \|_2 \,. \tag{4.14}$$

The last concept used aims to defend **against duty cycled attacks**, where an attacker chooses to inject a periodic attack signal with breaks in between repetitions consisting of

Algorithm 6: The proposed secure state estimator per sensor group G with $p_a > 1$

Step 1: Collect multiple $\mathbf{y}_i$ from Algorithm 4 to fill the measurements window $\mathcal{N}_i$.
Step 2: Get appropriate initial point $\mathbf{x}_{\text{init}}$ for the optimizer for time step i and initialize $\mathbf{\Omega}$ if $i == 0$.
if $i == 0$ **then**,
  Go to Algorithm 5.
else
  if $|\sigma_{\mathcal{N}_i} - \sigma_{\mathcal{N}_{i-1}}| > \sigma_{\tau_1}$ **then**
    $\mathbf{x}_{\text{init}} = \mathbf{0}_p$
  else
    $\mathbf{x}_{\text{init}} = \hat{\mathbf{x}}_{i-1}$
  end if
end if
Step 3: Solve one of the following optimization problems:
if $\sigma_{\mathcal{N}_i} > \sigma_{\tau_2}$ **then**

$$\hat{\mathbf{x}}_i = \underset{\mathbf{x}_i}{\operatorname{argmin}} \sum_{j \in \mathcal{N}_i} \|\mathbf{\Omega}^{:,j} \bullet \mathbf{y}_j - \mathbf{H}\,\mathbf{x}_i\|_2$$
$$\text{subject to } \mathbf{x}_{\text{lb}} < \mathbf{x}_i < \mathbf{x}_{\text{ub}}$$

else

$$\hat{\mathbf{x}}_i = \underset{\mathbf{x}_i}{\operatorname{argmin}} \sum_{j \in \mathcal{N}_i} \|\mathbf{y}_j - \mathbf{H}\,\mathbf{x}_i\|_2 + \lambda\|\mathbf{a}_i\|_1$$
$$\text{subject to } \mathbf{x}_{\text{lb}} < \mathbf{x}_i < \mathbf{x}_{\text{ub}}$$

end if

no attacks but only sensor readings. This attempt shall fool the estimator and ultimately yield in an undesired state. ReCaP aims to detect such periods (i.e. a window suffering a smart attack and one which does not), by taking a look at the standard deviation $\sigma_{\mathcal{N}_i}$ of the window $\mathcal{N}_i$ and comparing it to a (small) threshold σ_{τ_2}.

In the case that all sensors agree on the same reading within the noise bounds[4], this threshold is not met, indicating that the sensors are currently not under attack. Therefore ReCaP can utilize a different optimization problem in an attempt to enhance the estimation results. Hence, ReCaP solves either of the two optimization problems (solved in Algorithm 6)

$$\hat{\mathbf{x}}_i = \begin{cases} \operatorname{argmin}_{\mathbf{x}_i} \sum_{j \in \mathcal{N}_i} \|\mathbf{\Omega}^{:,j} \bullet \mathbf{y}_j - \mathbf{H}\,\mathbf{x}_i\|_2, & \text{if } \sigma_{\mathcal{N}_i} > \sigma_{\tau_2} \\ \operatorname{argmin}_{\mathbf{x}_i} \sum_{j \in \mathcal{N}_i} \|\mathbf{y}_j - \mathbf{H}\,\mathbf{x}_i\|_2 + \lambda\|\mathbf{a}_i\|_1, & \text{else} \end{cases}. \tag{4.12}$$

[4]Note that this could also mean that all sensors suffer an attack which simply adds the same constant (i.e. $\mathbf{a}_i = [c, c, ..., c]^{\mathrm{T}}$ with $c \in \mathbb{R}\backslash 0$), in this case not enough side information is available to extract the true value of the readings. In other words, this constitutes the **impossible problem** for which there currently exists no solution to securely estimate the state correctly.

Further, since Algorithm 4 may only forwards a **single reading** to the second stage ($p_{\mathrm{a}} = 1$ but $p > 1$) or because $p = p_{\mathrm{a}} = 1$, it is necessary to provide an alternate formulation to Algorithm 6. In this case, Algorithm 7 is deployed, as there is only a single sensor reading available, there is no need for rotating weights, however, multiple sensors are simulated, by dividing the long window w in sub windows of size w_{sub} on which the perturbations are applied. Note that in case of an attack of type B, the attacker may choose to apply an attack that results in $p_{\mathrm{a}} = 0$, in this case ReCaP cannot guarantee an acceptable estimation error and this is considered a total denial of service attack.

Algorithm 7: Pseudo-code of the proposed secure state stator per sensor group G with $p_a = 1$

Step 1: Collect multiple $\mathbf{y}_i$ from Algorithm 4 to fill the measurements window $\mathcal{N}_i$. The size of $\mathbf{y}_i$ would be one as $p_a = 1$.

Step 2: Divide the long window w of measurement to sub-windows of size $\mathrm{w_{sub}}$ where $\mathrm{w_{sub}} << \mathrm{w}$.

$\mathbf{tmp}_j = [\mathbf{y}_i; \mathbf{y}_{i+1}; ...; \mathbf{y}_{i+\mathrm{w_{sub}}}]$

$\mathbf{y}_j = \mathbf{tmp}$;

Step 3: Adapt the algorithm parameters like $\mathbf{H}$ to the new size of measurements by combining multiple in a column.

Step 4: Get appropriate initial point $\mathbf{x}_{\mathrm{init}}$ for the optimizer for time step i and initialize $\mathbf{\Omega}$ if $i == 0$.

if $i == 0$ **then**,

Go to Algorithm 5.

else

if $|\sigma_{\mathcal{N}_i} - \sigma_{\mathcal{N}_{i-1}}| > \sigma_{\tau_1}$ **then**

$\mathbf{x}_{\mathrm{init}} = \mathbf{0}_p$

else

$\mathbf{x}_{\mathrm{init}} = \mathbf{x}_{i-1}$

end if

end if

Step 5: Solve one of the optimization problems:

if $\sigma_{\mathcal{N}_i} > \sigma_{\tau_2}$ **then**

$$\hat{\mathbf{x}}_i = \underset{\mathbf{x}_i}{\operatorname{argmin}} \sum_{j \in \mathcal{N}_i} \|\mathbf{\Omega}^{:,j} \bullet \mathbf{y}_j - \mathbf{H}\,\mathbf{x}_i\|_2$$

$$\text{subject to: } \mathbf{x}_{\mathrm{lb}} < \mathbf{x}_i < \mathbf{x}_{\mathrm{ub}}$$

else

$$\hat{\mathbf{x}}_i = \underset{\mathbf{x}_i}{\operatorname{argmin}} \sum_{j \in \mathcal{N}_i} \|\mathbf{y}_j - \mathbf{H}\,\mathbf{x}_i\|_2 + \lambda \|\mathbf{a}_i\|_1$$

$$\text{subject to: } \mathbf{x}_{\mathrm{lb}} < \mathbf{x}_i < \mathbf{x}_{\mathrm{ub}}$$

end if

4.3. Robustness of Attack-resilient State Estimators (RARSE)

Pajic et al. [45] propose a secure CPS algorithm not only on the assumption of noise (and attacks) but also aim to tackle the problem of a not perfectly accurate system model. The main restriction of this work is, that it deals with a linear system model; therefore the linear localization described in Section 2.2.3 will be used to *fairly* compare it with the other two approaches discussed.

4.3.1. Problem statement

This approach considers a Linear-Time Invariant (LTI) system in the form

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{A}\,\mathbf{x}_k + \mathbf{B}\,\mathbf{u}_k + \mathbf{v}_i \\ \mathbf{y}_k &= \mathbf{C}\,\mathbf{x}_k + \mathbf{w}_k + \mathbf{e}_k \end{aligned}, \tag{4.10}$$

where $\mathbf{x}_k \in \mathbb{R}^n$ and $\mathbf{u}_k \in \mathbb{R}^m$ correspond to the systems state and input vector at time k, respectively. Further, $\mathbf{y}_k \in \mathbb{R}^p$ corresponds to the measurements taken from the set of p sensors $\mathcal{S} = \{s_1, \dots, s_p\}$. Additionally, $\mathbf{v}_k \in \mathbb{R}^n$ and $\mathbf{w}_k \in \mathbb{R}^p$ model process- and measurement noise, respectively, while $\mathbf{e}_k$ represents the attack vector at time k. It is assumed that the sensors included in the set $\mathcal{K} = \{1, 2, \dots, p\}$ are under attack, therefore $\mathbf{e}_{k,i} = 0 \,\forall\, \mathcal{K}^{\mathcal{C}}$ and $k \geq 0$. Note that $\mathcal{K}^{\mathcal{C}} = \mathcal{S} \backslash \mathcal{K}$, therefore $\text{supp}\,(\mathbf{e}_k) \subseteq \mathcal{K}\,, \forall\, k \geq 0$.

It has been shown in [46], that the optimal secure estimate of (4.10) – in the lack of noise, i.e. $\mathbf{v}_k = \mathbf{0}_n$ and $\mathbf{w}_k = \mathbf{0}_p$ – is given by

$$\hat{\mathbf{x}} = \underset{\hat{\mathbf{x}} \in \mathbb{R}^n}{\text{argmin}} \| \mathbf{Y}_{t,N} - \boldsymbol{\phi}_N\,(\mathbf{x}) \|_{l_0}\,, \tag{4.11}$$

utilizing the last N measurements $(\mathbf{y}_{t-N+1}, \dots, \mathbf{y}_t)$ and actuator inputs $(\mathbf{u}_{t-N+1}, \dots, \mathbf{u}_{t-1})$. Further, the matrix $\mathbf{Y}_{t,N} \in \mathbb{R}^{p \times N}$, formed by

$$\mathbf{Y}_{t,N} = \left[\tilde{\mathbf{y}}_{t-N+1} \middle| \tilde{\mathbf{y}}_{t-N+2} \middle| \dots \middle| \tilde{\mathbf{y}}_t \right]\,,$$

represents the collection of the last N sensor measurements with

$$\begin{aligned} \tilde{\mathbf{y}}_k &= \mathbf{y}_k\,, && \text{for } k = t - N + 1 \\ \tilde{\mathbf{y}}_k &= \mathbf{y}_k - \sum_{i=0}^{k-t+N-2} \mathbf{C}\,\mathbf{A}^i\,\mathbf{B}\;\mathbf{u}_{k-1-i}\,, && \text{for } k \geq t - N + 2 \end{aligned}. \tag{4.12}$$

Additionally, the system's evolution over the last N steps is captured in the linear mapping $\boldsymbol{\phi}_N\,(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}^{p \times N}$ which is defined as

$$\boldsymbol{\phi}_N(\mathbf{x}) = \left[\mathbf{C}\,\mathbf{x} \middle| \mathbf{C}\,\mathbf{A}\,\mathbf{x} \middle| \dots \middle| \mathbf{C}\,\mathbf{A}^{N-1}\,\mathbf{x}\right] .$$

Finally, the history of the last N attacks is represented by $\mathbf{E}_{t,N} = \mathbf{Y}_{t,N} - \boldsymbol{\phi}_N(\mathbf{x}) \in \mathbb{R}^{p\times N}$, with

$$\mathbf{E}_{t,N} = \left[\mathbf{e}_{t-N+1} \middle| \mathbf{e}_{t-N+2} \middle| \dots \middle| \mathbf{e}_t\right] .$$

Therefore, entries equal to zero in $\mathbf{E}_{t,N}$ correspond to non attacked sensors at the corresponding time steps, while zero rows indicate that a sensor has not been attacked in the observed time frame. Furthermore, the estimator shown in (4.10) can recover the system's state after N time steps when a maximum of q sensors are under attack, if and only if (iff) $\forall\,\mathbf{x} \in \mathbb{R}\backslash 0$

$$\left|\mathrm{supp}\left(\mathbf{C}\,\mathbf{x}\right) \cup \mathrm{supp}\left(\mathbf{C}\,\mathbf{A}\,\mathbf{x}\right) \cup \dots \cup \mathrm{supp}\left(\mathbf{C}\,\mathbf{A}^{N-1}\,\mathbf{x}\right)\right| > 2q\,. \tag{4.13}$$

Lastly, the maximum number of attacked sensors, such that the state $\mathbf{x}$ can be recovered, is denoted by $q_{\max}$, which depends on N, $\mathbf{A}$ and $\mathbf{C}$. Hence, if $q \leq q_{\max}$, the minimal l_0 norm of (4.11) of a noiseless system is equal to q.

4.3.2. Attack-resilient state estimator

Real systems usually suffer from noise and, in some cases, modeling errors, hence, due to the simplified assumptions, the secure state estimator from (4.11) can only be used in a very limited manner. For example, if the noise terms indicate an attack on more than $q_{\max}$ sensors, the condition for correct operation is violated.

For simplicity reasons it is assumed that $\mathbf{u}_k = \mathbf{0}_m$ for $k \geq 0$ in the remainder of this section. Further the subscripts from $\mathbf{Y}_{t,N}$, $\mathbf{E}_{t,N}$ and $\boldsymbol{\phi}_N(\mathbf{x})$ are dropped as the case $t = N-1$ is considered, meaning that $\mathbf{x}_0$ is estimated. It is further assumed that $\left|\mathcal{K}\right| \leq q_{\max}$ and $\left|\mathbf{w}_k\right| \preceq \epsilon_{w_k}$ as well as $\left|\mathbf{v}_k\right| \preceq \epsilon_{v_k}$ for $k = 0, \dots, N-1$. Therefore, the matrix

$$\mathbf{Y}_{\mathbf{w},\mathbf{v}} = \left[\mathbf{y}_0 \middle| \mathbf{y}_1 \middle| \dots \middle| \mathbf{y}_{N-1}\right] ,$$

contains the measurements including noise. Lastly, the noiseless version of $\mathbf{Y}_{\mathbf{w},\mathbf{v}}$ is given by

$$\bar{\mathbf{Y}} = \left[\bar{\mathbf{y}}_0 \middle| \bar{\mathbf{y}}_1 \middle| \dots \middle| \bar{\mathbf{y}}_{N-1}\right] .$$

This allows to define the optimization problem

$$\begin{aligned} P_0(\mathbf{Y}): \qquad & \min_{\mathbf{E},\mathbf{x}} \|\mathbf{E}\|_{l_0} \\ \text{s.t.:} \qquad & \mathbf{E} = \mathbf{Y} - \boldsymbol{\phi}(\mathbf{x}) \end{aligned} , \tag{4.14}$$

therefore

$$(\mathbf{x}_0, \mathbf{E}) = \text{argmax } P_0\left(\bar{\mathbf{Y}}\right) , \tag{4.15}$$

with $q = \|\mathbf{E}\|_{l_0} \leq q_{\max}$. Since the actual system does not have access to the ideal, noiseless, matrix $\bar{\mathbf{Y}}$, a relaxation of (4.14) is necessary to be able to deal with $\mathbf{Y}_{\mathbf{w},\mathbf{v}}$. This is achieved by

$$\begin{aligned} P_{0,\boldsymbol{\Delta}}\left(\mathbf{Y}\right): \qquad & \min_{\mathbf{E},\mathbf{x}} \|\mathbf{E}\|_{l_0} \\ \text{s.t.:} \qquad & |\mathbf{Y} - \boldsymbol{\phi}\left(\mathbf{x}\right) - \mathbf{E}| \preceq \boldsymbol{\Delta} \end{aligned} , \tag{4.16}$$

where $\boldsymbol{\Delta} \in \mathbb{R}^{p \times N}$ contains the tolerances $\delta_{j,i}$ for each sensor s_i, $i = 1, \ldots, p$, i.e.

$$\boldsymbol{\Delta} = \left[\boldsymbol{\delta}_0 \middle| \boldsymbol{\delta}_1 \middle| \ldots \middle| \boldsymbol{\delta}_{N-1}\right] .$$

Hence

$$\begin{aligned} \left(\mathbf{x}_{0,\boldsymbol{\Delta}}, \mathbf{E}_{0,\boldsymbol{\Delta}}\right) &= \text{argmax } P_{0,\boldsymbol{\Delta}}\left(\mathbf{Y}_{\mathbf{w},\mathbf{v}}\right) \\ q_{\boldsymbol{\Delta}} &= \|\mathbf{E}\|_{l_0} \end{aligned} . \tag{4.17}$$

Lastly, it is important to initialize $\boldsymbol{\Delta}$ appropriately, so that $P_{0,\boldsymbol{\Delta}}\left(\mathbf{Y}\right)$ has a feasible point $\left(\mathbf{x}_{0,\boldsymbol{\Delta}}, \mathbf{E}_{0,\boldsymbol{\Delta}}\right)$ such that $\|\mathbf{E}\|_{l_0} \leq q_{\max}$. To achieve this, the bound

$$\begin{aligned} |\mathbf{y}_k - \bar{\mathbf{y}}_k| &\leq |\mathbf{C}| \sum_{i=0}^{k-1} |\mathbf{A}^{k-1-i}| \, |\mathbf{v_i}| + |\mathbf{w}_k| \\ &\leq |\mathbf{C}| \sum_{i=0}^{k-1} |\mathbf{A}^{k-1-i}| \, \epsilon_{v_i} + \epsilon_{w_k} = \boldsymbol{\delta}_k , \end{aligned} \tag{4.18}$$

is specified. Therefore, there exists a feasible point $\left(\mathbf{x}_{0,\boldsymbol{\Delta}}, \mathbf{E}_{0,\boldsymbol{\Delta}}\right)$ for $P_{0,\boldsymbol{\Delta}}\left(\mathbf{Y}\right)$ if $\boldsymbol{\delta}_k \succeq \bar{\boldsymbol{\delta}}_k$ for $k = 0, \ldots, N-1$. Meaning that $q_{\boldsymbol{\Delta}} = q \leq q_{\max}$ and a solution can be obtained if at most $q_{\max}$ sensors are compromised.

Lastly, Pajic et al. provide an algorithm to calculate a bound in which the estimated state lies. As this will not be used in the implementation related to this thesis, it is not further discussed here; the interested reader is directed to [45].

5. Analysis

The flexibility and potentials of the proposed framework, introduced in Chapter 3, are now studied by deploying the algorithms described in Chapter 4 in different test cases which are discussed before presenting the results obtained by each secure state estimator.

5.1. Test cases

The different scenarios presented in this section evolve in complexity and aim to stress both, the simulation environment and the secure state estimation algorithms. As the overall simulation environment, OpenUAV, deals – as explained – with UAVs as a simulated robot, all test cases involve a number of UAVs.

It is important to note, before going into details, that the measured signals (which may be corrupted) are directly or indirectly used as input for the flight controllers[1] which control the actuators, and ultimately the position of the UAV. Therefore, a smart attack on an unsecured system can be used to maliciously drive the UAV to any position the adversary would like it to go; or even crash. All range measurements conducted in this analysis make use of a simulated UWB transceiver based on [41].

5.1.1. Test case 1

The first scenario consists of a single UAV, equipped with a laser range finder and a wireless module, to measure the distance between the ground and eight anchor nodes at known locations, respectively. The environment is contained within a $5\,\mathrm{m} \times 5\,\mathrm{m}$ plane, while the target position is at the center $(2.5, 2.5)$ and shall be kept at all times (as can be seen in Figure 5.1).

As described in Chapter 2, the localization is carried out in 2D. This results in a total of 8 links (8 anchors), all of which are considered under attack by an attacker following both, type A and type B schemes.

[1] The flight controllers make use of various estimators for estimating its current status (altitude, position, etc.), a P-loop controller for reducing the positioning error and a PID-loop controller for reducing the velocity error in order to reach the target destination quickly and correctly. For details please refer to [34].

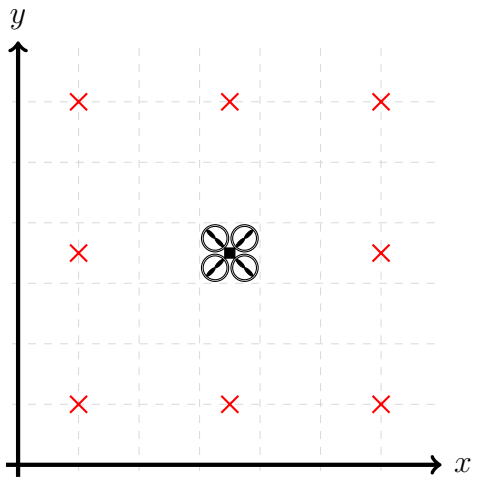


Figure 5.1.: Setup for the first test case, consisting of one UAV in the center, surrounded by eight anchors (red crosses).

5.1.2. Test case 2

In this setup, two UAVs are deployed in the same conditions as in test case one, while one UAV still tries to remain at a given point in the environment, a second constantly measures its own position and distance to the first UAV and tries to stay at a certain distance from it (c.f. Figure 5.2).

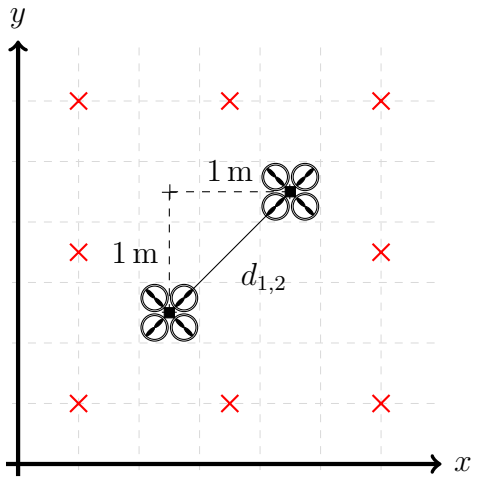


Figure 5.2.: Setup for the second test case, consisting of one UAV in the top right corner, another one in the bottom left, constantly monitoring the distance between both and trying to keep at a target distance from the first UAV, both are surrounded by eight anchors (red crosses).

Therefore, on top of the 8 links identified in the first test case, 9 more – totaling 17 links – are considered under attack in this scenario. In addition to the 8 links necessary to control the location of the second UAV, one additional link is required for measuring the distance between both UAVs.

5.2. Results

All use-cases have been implemented and the secure estimators (introduced in Chapter 4) were deployed in separate scenarios. While the attack signal on the sensors is not the same for all simulations, the statistical characteristics is the same (e.g. the same shift and variance values during the same periods), the attack signal used can be seen in Figure 5.3.

Here, for the first 200 time steps, the adversary chooses to add Gaussian noise with a mean of 3 m and a variance of 2 m to the measured ranges between the anchors and the UAV(s), the next 200 time steps are not subject to an attack in order to confuse the state estimator, which is followed by a Gaussian noise attack with mean 4 m and a variance of 2 m in the next 200 time steps; in the last 200 time steps, the attack has a mean of 2 m and a variance of 5 m. For the attacks on the range measurement between the UAVs, the means are 0.5 m, 0 m, 0.9 m and 0.5 m, all with a variance of 2 m for the same periods, respectively. The different mean values for the attack on the range measurement between the two UAVs have been chosen because the initial position of both UAVs is relatively close to each other, therefore it is not expected to make use of the whole simulation area. The attack results in a few outliers which are filtered out by the intrusion detector before, if an algorithm makes use of such.

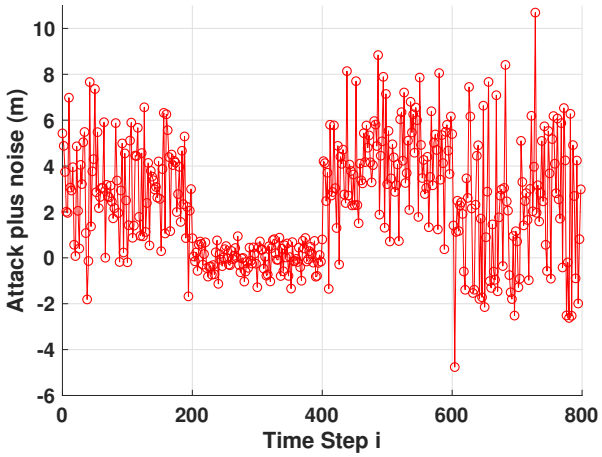


Figure 5.3.: The attack being applied over time on all sensors being considered under attack.

The resulting absolute errors in terms of estimated position (test case 1 and 2) as well as of the estimated range between the two UAVs (use case 2) are shown in Figures 5.4-5.6 and are defined as

$$\mathrm{err}_{\mathrm{pos},i} = \|\hat{\mathbf{p}}_i - \mathbf{p}_i\|_2 \,, \tag{5.1}$$

$$\mathrm{err}_{\mathrm{rng}} = |\hat{r} - r| \,, \tag{5.2}$$

where $\mathbf{p}_i$ and $\hat{\mathbf{p}}_\mathbf{i}$ are the real and estimated position of UAV $i \in [1, 2]$. Further, $\hat{r}$ and r represent the measured and actual range between the two UAVs, respectively.

Table 5.1.: Comparison of the three secure state estimators in terms of absolute errors and computation time. Note that the timing information provided always corresponds to a single execution (i.e one sensor group in ReCaP, one time step in all other algorithms).

Test Case	Metric	SecEKF	SecOPT	ReCaP	RARSE
1	Mean error	0.065	0.2965	0.0356	0.8613
	Std. Deviation	0.062	0.451	0.0211	0.1453
	Computation time	246.07 µs	28.4 ms	56.91 ms	540.5 ms
2	Mean error UAV_1	0.1129	0.2079	0.0369	0.7749
	Std. Deviation UAV_1	0.0596	0.1428	0.0177	0.0522
	Mean error UAV_2	0.0745	0.2222	0.2893	0.5092
	Std. Deviation UAV_2	0.044	0.1466	0.1802	0.0668
	Computation time	568.06 µs	134.35 ms	59.21 ms	1.31 s

The mean and variance of the absolute errors as well as the mean time needed for computing an estimate for one time step is summarized in Table 5.1.

It can be seen that all three algorithms perform reasonably well over the whole course of the simulations, before directly and *fairly* comparing them with each other, the results will be discussed individually to highlight advantages and disadvantages of each secure state estimator.

It has to be highlighted that the focus of this work was not on high performance implementations, therefore, any optimization-based algorithm does not meet the real-time requirements set by OpenUAV. However, if computations would rely on better solvers (currently standard SciPy solvers [47] are used) and make use of parallelization (while running on an overall faster system), these requirements can be met (as has been demonstrated by [36], for example).

5.2.1. SecSens

Figure 5.4 shows the corresponding performance of SecSens, as it is a collection of two individual algorithms, the discussion is split in two parts as well.

5.2.1.1. SecEKF

The clear advantage of SecEKF over any other considered secure state estimator is, that it is soley based on simple matrix operations which, in the studied context, do not constitute as a "big" problem (i.e. several hundred states to be estimated). Therefore, it is the fastest algorithm simulated during this work. Furthermore, this fact allows SecEKF to be deployed directly into OpenUAV without heavy optimization w.r.t. computation time. The resulting absolute errors remain low throughout the tests and the attacks are filtered out efficiently. It can be seen that it performs equally good in both test cases, indicating

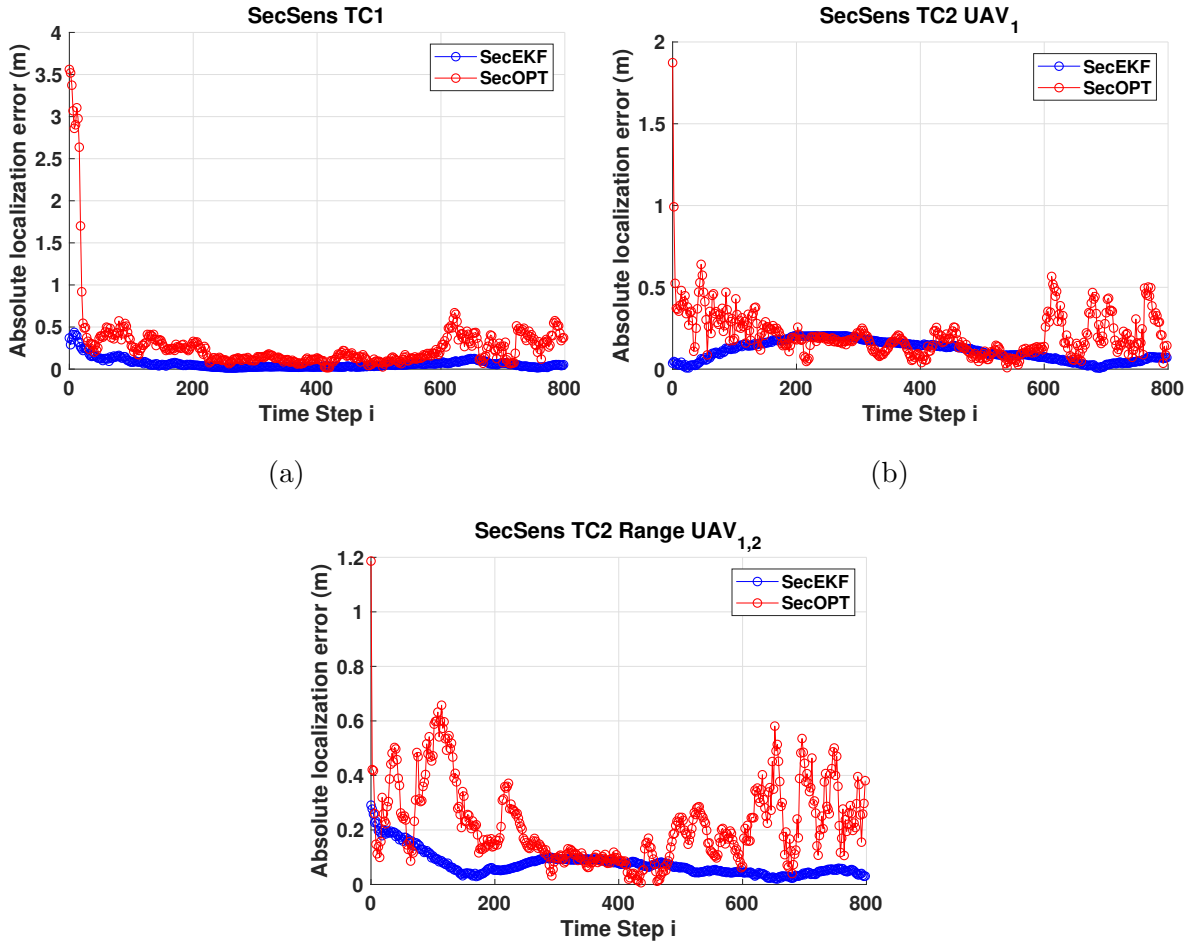


Figure 5.4.: Simulation results for SecEKF (blue) and SecSens (red), the absolute localization error for test case 1 (a) and the absolute errors in terms of the location of UAV_1 (b) and the range in between UAV_1 and UAV_2 (c) all show excellent performance.

that the number of links (i.e. measurements) and states does not yield in degraded performance, at least in the small scale.

5.2.1.2. SecOPT

Similar to SecEKF, SecOPT performs well within the expected boundaries for a localization problem. Due to its nature (optimization based), the run-time suffers as the implementation was not optimized for speed and standard solvers have been used. Therefore, the secure estimates have been carried out offline by firstly collecting the measurements from OpenUAV, attacking them and feeding SecOPT with them. Afterwards, the secure estimates have been used as new way points for a second simulation in OpenUAV, showing its performance. This has been done for both test cases.

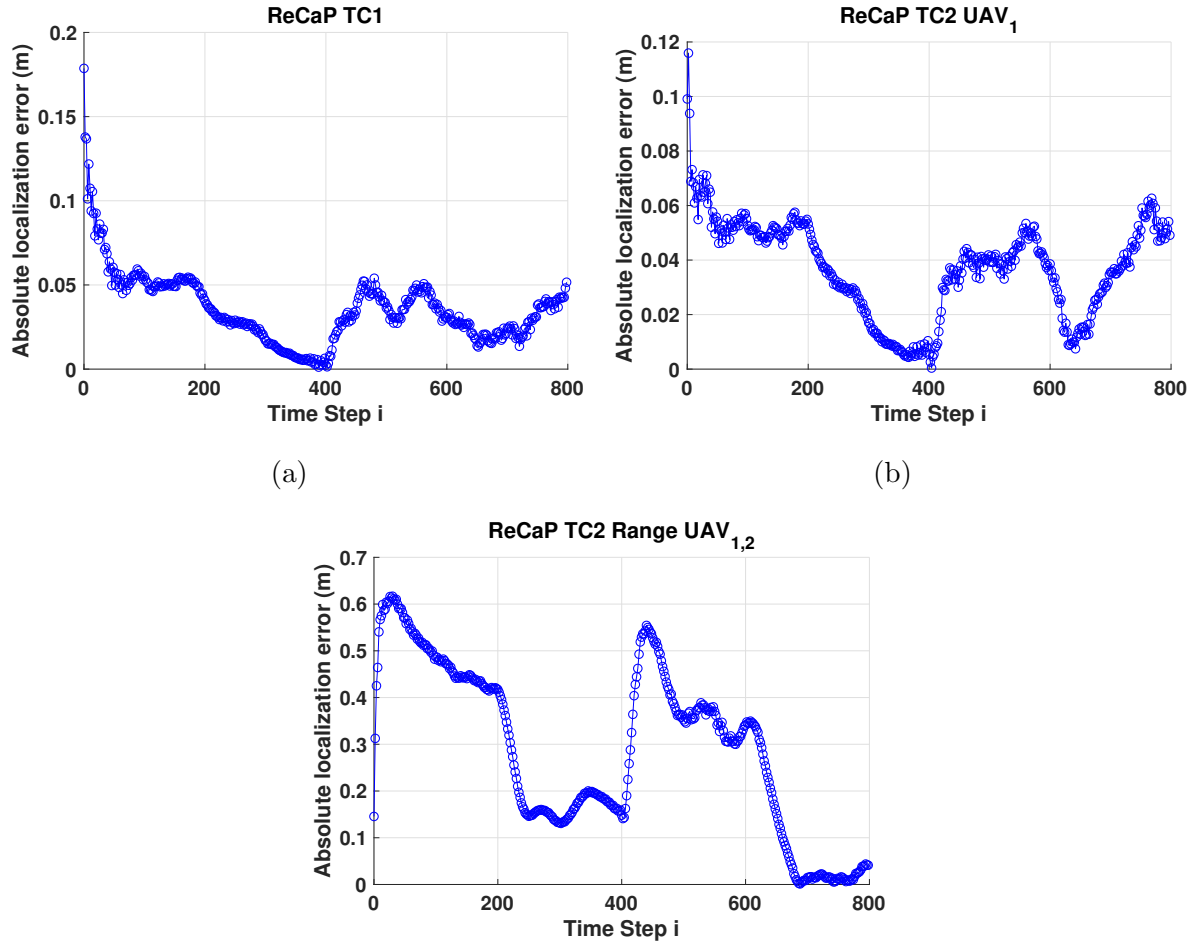


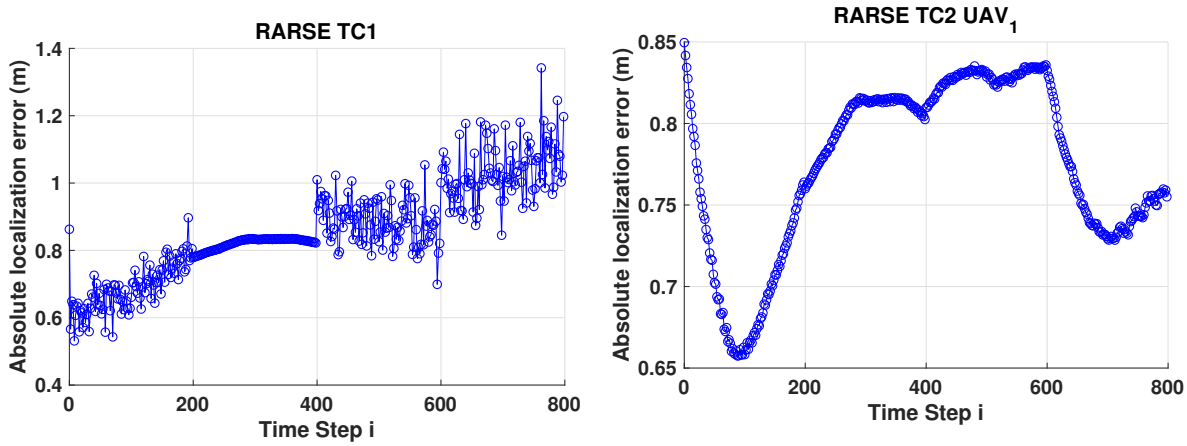
Figure 5.5.: Simulation results for ReCaP, the absolute localization error for test case 1 (a) and the absolute errors in terms of the location of UAV_1 (b) and the range in between UAV_1 and UAV_2 (c) all show excellent performance.

5.2.2. ReCaP

The big advantage of ReCaP is the effort needed to implement it on top of an, already in place, industrial firmware. In this specific case, the underlying firmware was an EKF which estimates the position(s) of the UAVs based on the secure measurements provided from ReCaP. As every sensor group needs its own instance of ReCaP, the computation time suffers (m vs 1 optimization problems need to be solved each time step). The overall performance however, is – again – within reasonable bounds for these problems (c.f. Figure 5.5).

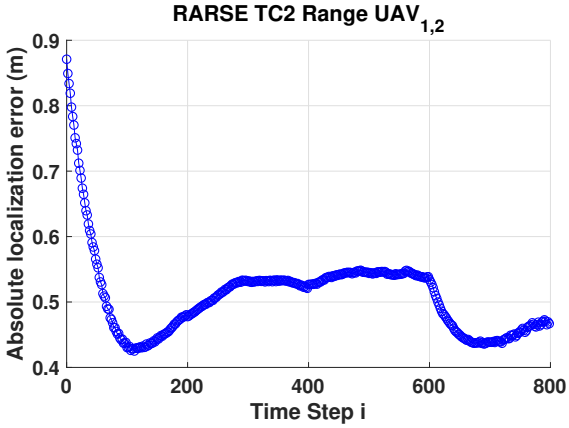
5.2.3. RARSE

Lastly, the attack-resilient state estimator has been deployed for both test cases. As it cannot handle non-linear problems, the linearization technique described in Chapter 2 has been used to linearize the problem. As this linearization is not a perfect solution and,



(a)

(b)



(c)

Figure 5.6.: Simulation results for RARSE, the absolute localization error for test case 1 (a) and the absolute errors in terms of the location of UAV_1 (b) and the range in between UAV_1 and UAV_2 (c) all show excellent performance.

in itself, subject to error, the absolute errors for this secure state estimator are higher as well, but still well within any boundaries (see Figure 5.6). Again, the estimation utilizes an optimization based method, therefore computations have been carried out offline. It has to be noted, that not all links have been under attack, as RARSE has an upper limit ($q_{\max}$) on the number of attacked links.

5.3. Fair comparison

As can be seen in Figures 5.4-5.6 and Table 5.1, all implementations operate similarly well, considering their limitations (i.e. optimization based and linearization). Naturally, SecEKF provides the fastest computation time, while the non-optimized version of RARSE takes the longest time for providing secure estimates. Nevertheless, in terms of implementation effort, ReCaP clearly stands out as the underlying system does not need to be modified. This man-in-the-middle type of deployment enables it to be used in between

the sensor(s) and firmware allowing rapid and simple integration. Further, SecSens and ReCaP perform approximately equally good in terms of mean error and standard deviation, while RARSE performs slightly worse in all discussed cases. However, it has to be considered that the latter suffers from the disadvantage of having to deal with a non linear system it has not been designed to operate with. Keeping this in mind, it still performs comparably well to the other algorithms, therefore indicating that, if all would operate on a linear system model, it could perform just as good as the remaining secure state estimators. The big disadvantage of RARSE, compared to other algorithms is, that it cannot deal with an arbitrary high number of attacked links.

Ultimately, it can be seen that each implementation has its own set of advantages and disadvantages based upon the observed system and problem, while this could have been said based solely on the published works by the authors, SecUAV provides a unique opportunity to evaluate their performance on exactly the same problem. This enables tweaking of the implementations to see if the results would change (e.g. as discussed in the beginning of Section 5.2, focusing on parallelization and faster solvers).

6. Conclusion

In this work, the problem of fair comparison of secure CPS state estimation algorithms has been considered and a unified framework has been proposed. The newly introduced SecUAV – based on OpenUAV – enables students, researchers and industry alike, to rapidly compare multiple algorithms in a uniform setting. Therefore, allowing a fair comparison of different solutions to the same problem, before ultimately making a choice for a specific problem in mind. The capabilities of this framework have been demonstrated, by deploying three different algorithms on the same set of problems under similar conditions. It has been shown that the proposed framework is a suitable environment for simulating secure state estimators. Special care has to be taken however, if the algorithm is not centralized, in tweaking the execution time, so that normal operation of the UAV(s) is guaranteed.

Future work will focus on the completion of SecUAV as well as a fair comparison of, at the time of completion, state of the art secure state estimators. Furthermore, it has been shown that standard solvers of SciPy are not suitable for real time operation of the studied algorithms, in an attempt to provide samples allowing rapid learning of using the tool, optimized versions of the implemented algorithms will be made. Furthermore, the possibility of a challenge, similar to the NSF Student CPS Challenge, could be possible in the future (based on this framework), where researchers and students try to solve a given problem with their own secure state estimators competing for the most robust system.

Appendices

A. A deep dive into OpenUAV

A.1. ROS, Gazebo, PX4 and Docker

A.1.1. Robot Operating System

ROS [29] is an open-source collection of libraries, conventions and tools meaning to provide one common, simple to use, framework for robotic platforms. It is maintained by a big community contributing their knowledge to the ROS ecosystem. In order to allow maximum flexibility in terms of robot configurations, ROS has been built in a modular and distributed design to enable usage among a wide variety of robots. The distributed design allows that individuals contribute modules which can be invoked and used in a given setup. Further, the modular design lets the decision on what to use up to the developer. If someone would rather use provided modules for some sensors but needs a custom implementation for the remaining sensors and actuators, implementing this and integrating it with the remaining robot is easily doable without much overhead.

This modularity is mainly achieved by sub-processes called nodes, each node may resemble one or more components of the robot (e.g. an actuator, sensor or camera) which can publish data to the remainder of the system via a topic, which can be thought of as a URI, allowing another node to subscribe to this topic and read the published data in real-time (c.f. Figure A.1). The orchestration of these nodes and the robot is achieved by a core system named Master. Furthermore, the data is published by means of messages, which are a standardized format for inter-node communication. One main advantage of this architecture is, that the user can control all components of a robot via a single interface (publishing/subscribing to different topics) rather than having to deal with a different API for each component as it is usually the case in embedded systems.

Furthermore, the MAVROS package [30] is used, which enables a higher abstraction level of nodes and provides an interface to MAVLINK, the link used for communicating with PX4. This allows the user to focus on the algorithmic implementation rather than having to take care of every little detail of controlling actuators and sensors. For example, the UAV can simply be controlled by publishing appropriate data to topics, ROS, MAVROS and PX4 take care of the rest (e.g. control loops of the actuators and sensors).

A.1.2. Gazebo

While ROS enables researchers to create a modular software for their robots, it does not allow to simulate the developed algorithms and procedures, Gazebo [31] was born out of early ROS versions meaning to provide a solution to this issue. Like ROS, Gazebo

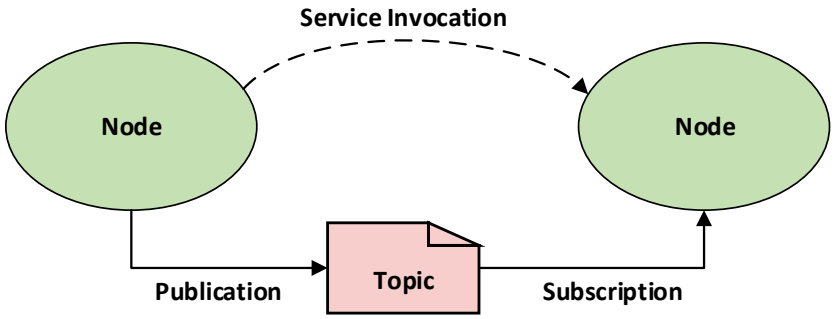


Figure A.1.: The basic architecture of ROS, a component can be represented by one or more nodes which publish and subscribe data to/from topics. The collection of nodes is orchestrated by the master.

is open-source and benefits from an active community, it combines a physics engine, graphical representation of the simulation as well as programmatic and graphical interfaces to control the simulation and its environment. Gazebo allows its users to create a world in which the robot moves, as well as to build a 3D model of the robot(s) being simulated. It further provides interfaces to simulate the sensors and actuators implemented in ROS so that they can be used in the simulation as well (e.g. cameras, laser range finders, LiDAR).

All of this combined allows to test algorithms and scenarios before they are being applied to the actual robot, allowing rapid development and debugging in a real-life like (regarding to physics) environment. As is the case with ROS, Gazebo was designed in a distributed manner, a master takes care of the orchestration while the physics engine, rendering, user interface, communication and sensor generation are being taken care of by separate libraries, which can be thought of like the concept of nodes described earlier. The integration of ROS with Gazebo is shown in Figure A.2, as can be seen the same ROS implementation can be used for real world tests as well as Gazebo simulations.

A.1.3. PX4

PX4 [32] is a commercial auto pilot for robots which, as the rest of the software stack, is open source and open-hardware. It allows the user to only have to take care of the "big picture" rather than every little detail. For example, the user specifies a set of way points to which the drone shall fly, once passed to PX4, the platform takes care of actuator control and takes sensor readings into account in order to avoid obstacles and to fly to the specified locations. As the previous platforms, PX4 benefits from an active community as well and offers full flexibility to modify and adapt it fully to the user's needs. The architecture (see Figure A.3) allows easy replacement of different blocks if they are not needed or if the user needs to modify a block in order to work with the target application. It further includes all necessary interfaces to control actuators and read sensors so that the mission can be successfully finished.

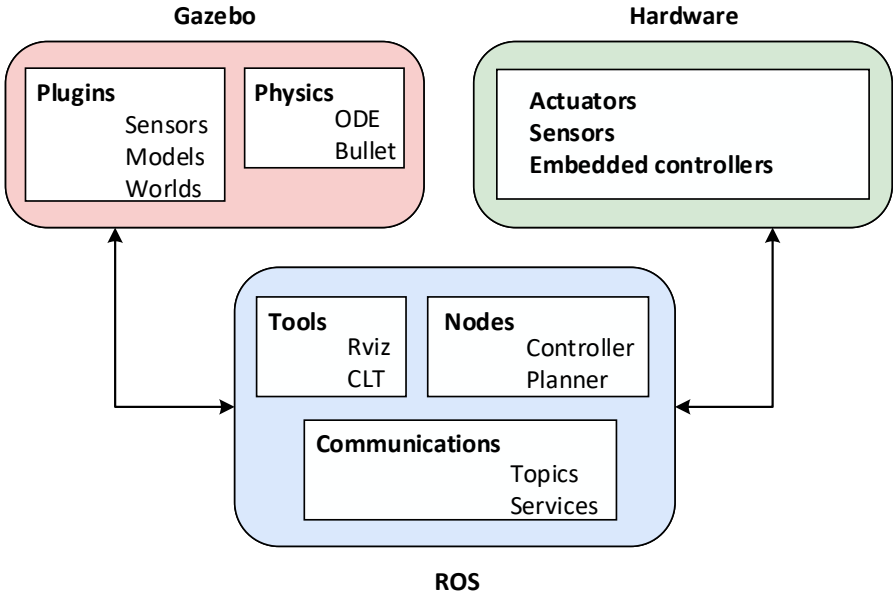


Figure A.2.: Integration of ROS and Gazebo, ROS can be simulated by Gazebo before being deployed to the real world.

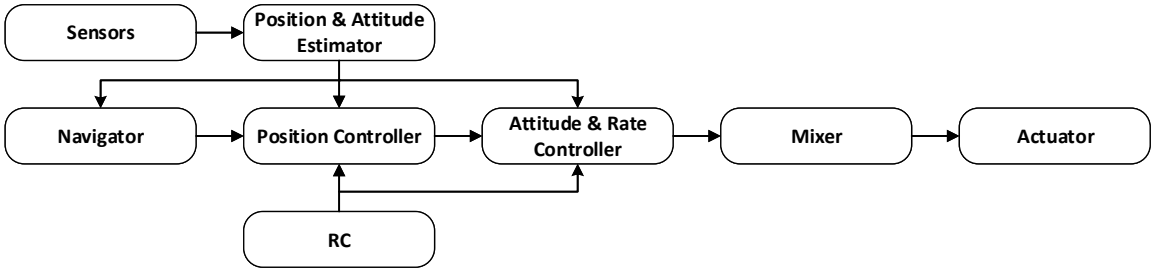


Figure A.3.: The architecture of the PX4 flight stack.

A.1.4. Docker

Docker [33] is a container platform that enables organizations and users to rapidly build and deploy services, platforms and more in containers. A basic use case is, among others, Container as a Service, in this case the user is provided with a script (dockerfile) which, when executed, creates everything necessary for the service to operate on a local Docker ecosystem. As can be seen in Figure A.4, Docker takes care of governance, security, authorization and orchestration of deployed containers. A container itself can be a rather simple program running on it, or a complete operating system (e.g. Ubuntu) which in turn is used to launch a service or process which then communicates with the remaining containers or the outside world. This architecture allows rapid scaling and prototyping of services, if for example, more instances of a service are needed, all that needs to be done is to deploy more containers. The scaling – among other features – can be automated by using Kubernetes, Ansible, or other container management software.

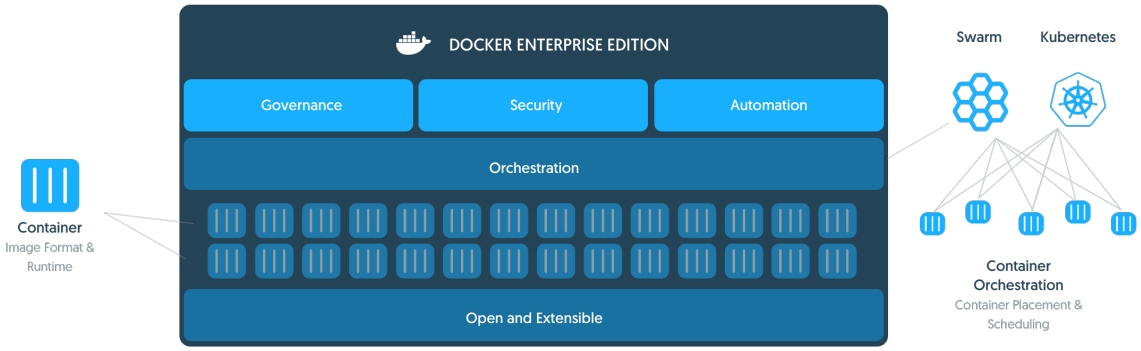


Figure A.4.: The architecture of Docker [48]. An application runs in a container which, in turn, is managed by the underlying Docker system, handling governance, security, automation and host integration.

A.2. Capabilities

To further strengthen the reasoning behind choosing OpenUAV over CARLA and Apollo as the basis of SecUAV, the flexibility and capabilities of OpenUAV are demonstrated by summarizing two case studies presented by the authors of [28] and showing the feasibility of OpenUAV for the intended use – sensory attacks.

A.2.1. Case study 1 - Machine Learning

Having been built with adaptivity in mind, OpenUAV can be extended, and make use of various computational platforms and frameworks such as openCV [49] and tensorflow [50] which are libraries for computer vision and machine learning, respectively. In order to show off these capabilities, Schmittle et al. created a simulation where an UAV has to navigate itself through a world with obstacles, such as trees, to reach a given point. This task has been achieved by utilizing openCV to compute a depth map from the cameras of the UAV which clearly shows any obstacles in the way of the drone. These images are then fed to tensorflow which uses them to calculate a set of instructions for OpenUAV to use to control the quadrotor into the right direction.

Therefore, it is possible to use external libraries in the simulation environment, enabling many possibilities for potential test cases while keeping the complexity of the solution low as not everything needs to be implemented by the user. The usage of openCV can be especially useful when relying on camera feeds to get certain information (e.g. ranges to objects) while tensorflow enables the community to further evaluate secure CPS algorithms utilizing deep learning.

A.2.2. Case study 2 - Formations

The second test case shown by the authors deals with the goal of having multiple UAVs fly in a given formation in a leader-follower scenario. Would a challenge like this be done using real hardware, it is likely for UAVs to crash into each other during development of the algorithm, yielding in a very costly implementation. Further it aims to show how easy it is to control multiple drones in a simulation, as well as the feature of utilizing MATLAB for computations needed by the simulation. In this simulation, MATLAB is used offline to compute a set of way points for each UAV which are then fed to the simulation environment letting each UAV fly to the specified points.

Offline computations can be necessary – where applicable – if either (i) the user already has an implementation in another context (e.g. MATLAB) and wishes to quickly test if the effort of implementing it as a node is feasible or (ii) the algorithm is too computationally expensive for yielding results in real-time on the UAV testbed (e.g. because it is a centralized algorithm which would usually run on a high performance back-end server).

A.2.3. NSF Student CPS Challenge

Further, OpenUAV is used during the initial phase of the NSF Student CPS Challenge [51], in which teams of students are presented with a problem in the field of UAV research and compete to solve it in a better way than other teams. Before the final field competition on real hardware, the students utilize OpenUAV to simulate their ideas and solutions to find a feasible approach to (hopefully) win the challenge. This shows how OpenUAV can be used in education to enable students to learn concepts previously only studied on paper or through very expensive hardware (if at all).

For the 2018 edition of the challenge, the teams were tasked with developing a system able to scan an area for a lost aircraft (i.e. another UAV) by using a quadrotor with downward facing camera and, if a team thinks it is required, other sensors. After locating the broken UAV in the field, the goal is to retrieve it and bring it back to base, safely. The teams were provided with OpenUAV as a simulation platform and support with hardware decisions. The first round of the competition was carried out between 10 teams competing against each other for the best simulation results in OpenUAV. Afterwards 6 teams were selected for real life outdoor trials before heading to the finals.

A.2.4. Secure sensing

Lastly, it shall be shown that OpenUAV can be used for the target of secure sensing and what problems are faced without a common secure CPS framework. As discussed in Section 3.2.1.2, it is possible to provide custom algorithms and sensors into a simulation by implementing new nodes into the system. This can, of course, be used to simulate a test case where an UAV tries to keep a safe distance from an obstacle, e.g. a wall or the ground, by relying on sensor readings (e.g. laser range finders).

Several nodes acting as sensors, a distance control algorithm, as well as a secure sensing algorithm and an attacker have been implemented (the scenario is shown in Figure A.5).

While the results are presented in Chapter 5, it can be seen that it is possible for researchers and developers to implement such a test case by using OpenUAV as a basis. If this is done without a unified approach, the result will be similar to the current state, where everyone has a different way of attacking and/or evaluating their algorithms, therefore not allowing for a *fair* comparison.

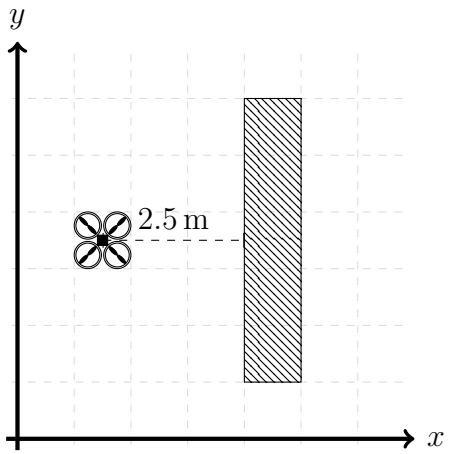


Figure A.5.: Secure sensing setup to show the capabilities of OpenUAV. The UAV is trying to consistently hover 2.5 m away from the obstacle (wall) by utilizing laser distance measurements.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] A. Alanwar, F. Miao, O. Ploder, J. Hespanha, and M. B. Srivastava, "ReCaP: Resilient Cyber Physical Systems Against Adversarial Sensor Attacks," Submitted to the 16th ACM Conference on Embedded Networked Sensor Systems.

[2] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," In *Computer Networks*, Vol. 52, No. 12, pp. 2292–2330, August 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128608001254

[3] U. Kiencke and L. Nielsen, *Automotive Control Systems: For Engine, Driveline, and Vehicle*. Springer Science & Business Media, December 2005, google-Books-ID: 8ayEIHoqvI4C.

[4] H. Heinecke, K. P Schnelle, H. Fennel, J. Bortolazzi, L. Lundh, J. Leflour, J. L Mat' E, K. Nishikawa, and T. Scharnhorst, "AUTomotive Open System ARchitecture-An Industry-Wide Initiative to Manage the Complexity of Emerging Automotive E/E-Architectures," 01 2004.

[5] M. Khurram, H. Kumar, A. Chandak, V. Sarwade, N. Arora, and T. Quach, "Enhancing connected car adoption: Security and over the air update framework," In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pp. 194–198, Dec 2016.

[6] R. Langner, "Stuxnet: Dissecting a Cyberwarfare Weapon," In *IEEE Security Privacy*, Vol. 9, No. 3, pp. 49–51, May 2011.

[7] D. Kushner, "The Real Story of Stuxnet," February 2013. [Online]. Available: https://spectrum.ieee.org/telecom/security/the-real-story-of-stuxnet

[8] M. H. Eiza and Q. Ni, "Driving with Sharks: Rethinking Connected Vehicles with Vehicle Cybersecurity," In *IEEE Vehicular Technology Magazine*, Vol. 12, No. 2, pp. 45–51, June 2017.

[9] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive Experimental Analyses of Automotive Attack Surfaces," In *Proceedings of the 20th USENIX Conference on Security*, ser. SEC'11, pp. 6–6. Berkeley, CA, USA: USENIX Association, 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=2028067.2028073

[10] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental Security Analysis of a Modern Automobile," In *2010 IEEE Symposium on Security and Privacy*, pp. 447–462, May 2010.

[11] J. P. Hubaux, S. Capkun, and J. Luo, "The security and privacy of smart vehicles," In *IEEE Security Privacy*, Vol. 2, No. 3, pp. 49–55, May 2004.

[12] M. Mohan, "Cybersecurity in drones - ProQuest," PhD Thesis, Utica College, 2016. [Online]. Available: https://search.proquest.com/docview/1796055579

[13] "Exclusive: Computer Virus Hits U.S. Drone Fleet." [Online]. Available: https://www.wired.com/2011/10/virus-hits-drone-fleet/

[14] A. H. Rutkin, "Yacht Captain Dares Researchers to Trick His GPS, Gets Unwelcome Result." [Online]. Available: https://www.technologyreview.com/s/517686/spoofers-use-fake-gps-signals-to-knock-a-yacht-off-course/

[15] A. J. Kerns, D. P. Shepard, J. A. Bhatti, and T. E. Humphreys, "Unmanned Aircraft Capture and Control Via GPS Spoofing," In *Journal of Field Robotics*, Vol. 31, No. 4, pp. 617–636. [Online]. Available: http://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21513

[16] A. A. Cárdenas, S. Amin, and S. Sastry, "Research Challenges for the Security of Control Systems." In *HotSec*, 2008.

[17] D. Martins and H. Guyennet, "Wireless Sensor Network Attacks and Security Mechanisms: A Short Survey," In *2010 13th International Conference on Network-Based Information Systems*, pp. 313–320, Sept 2010.

[18] D. F. Kune, J. Backes, S. S. Clark, D. Kramer, M. Reynolds, K. Fu, Y. Kim, and W. Xu, "Ghost Talk: Mitigating EMI Signal Injection Attacks against Analog Sensors," In *2013 IEEE Symposium on Security and Privacy*, pp. 145–159, May 2013.

[19] H. Fawzi, P. Tabuada, and S. Diggavi, "Secure estimation and control for cyber-physical systems under adversarial attacks," In *IEEE Transactions on Automatic Control*, Vol. 59, No. 6, pp. 1454–1467, 2014.

[20] A. J. Goldsmith and S. B. Wicker, "Design challenges for energy-constrained ad hoc wireless networks," In *IEEE Wireless Communications*, Vol. 9, No. 4, pp. 8–27, Aug 2002.

[21] G. J. Pottie and W. J. Kaiser, "Wireless Integrated Network Sensors," In *Commun. ACM*, Vol. 43, No. 5, pp. 51–58, May 2000. [Online]. Available: http://doi.acm.org/10.1145/332833.332838

[22] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A Survey of Computation Offloading for Mobile Systems," In *Mobile Networks and Applications*, Vol. 18, No. 1, pp. 129–140, February 2013. [Online]. Available: https://doi.org/10.1007/s11036-012-0368-0

[23] A. Alanwar, Y. Shoukry, S. Chakraborty, P. Martin, P. Tabuada, and M. Srivastava, "PrOLoc: Resilient Localization with Private Observers Using Partial Homomorphic Encryption," In *2017 16th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pp. 41–52, April 2017.

[24] A. Awad, T. Frunzke, and F. Dressler, "Adaptive Distance Estimation and Localiza-

tion in WSN using RSSI Measures," In *10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD 2007)*, pp. 471–478, Aug 2007.

[25] A. H. Sayed, A. Tarighat, and N. Khajehnouri, "Network-based wireless location: challenges faced in developing techniques for accurate wireless location information," In *IEEE Signal Processing Magazine*, Vol. 22, No. 4, pp. 24–40, July 2005.

[26] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An Open Urban Driving Simulator," In *Proceedings of the 1st Annual Conference on Robot Learning*, pp. 1–16, 2017.

[27] "Apollo." [Online]. Available: http://apollo.auto/

[28] M. Schmittle, A. Lukina, L. Vacek, J. Das, C. P. Buskirk, S. Rees, J. Sztipanovits, R. Grosu, and V. Kumar, "OpenUAV: A UAV Testbed for the CPS and Robotics Community," p. 9, 2018.

[29] "ROS.org | Powering the world's robots." [Online]. Available: http://www.ros.org/

[30] "mavros - ROS Wiki." [Online]. Available: http://wiki.ros.org/mavros

[31] "Gazebo." [Online]. Available: http://gazebosim.org/

[32] "PX4." [Online]. Available: http://px4.io/

[33] "Docker." [Online]. Available: https://www.docker.com/

[34] "OpenUAV Firmware." [Online]. Available: https://github.com/OpenUAV/Firmware

[35] "Welcome to the home page of the Cyber-Physical Systems Virtual Organization | CPS-VO." [Online]. Available: https://cps-vo.org/

[36] A. Alanwar, B. Etzlinger, H. Ferraz, J. P. Hespanha, and M. Srivastava, "SecSens: Secure State Estimation with Application to Localization and Time Synchronization," In *CoRR*, Vol. abs/1801.07132, 2018. [Online]. Available: http://arxiv.org/abs/1801.07132

[37] J. K. U. Simon J. Julier, "New extension of the Kalman filter to nonlinear systems," pp. 3068 – 3068 – 12, 1997. [Online]. Available: https://doi.org/10.1117/12.280797

[38] A. Alanwar, B. Etzlinger, H. Ferraz, J. Hespanha, and M. Srivastava, "SecSens: Secure State Estimation with Application to Localization and Time Synchronization," In *arXiv preprint arXiv:1801.07132*, 2018.

[39] A. Alanwar, N. Snyder, B. Etzlinger, S. Chakraborty, J. Hespanha, and M. Srivastava, "ReSeT: Reformulating Secure State Estimation for Resilient Cyber-Physical Systems," 2018.

[40] R. Ivanov, M. Pajic, and I. Lee, "Attack-resilient sensor fusion for safety-critical cyber-physical systems," In *ACM Transactions on Embedded Computing Systems (TECS)*, Vol. 15, No. 1, p. 21, 2016.

[41] "DecaWave DW1000 IR-UWB," http://www.decawave.com/products/dw1000, accessed: 2018-4-01.

[42] A. Alanwar, H. Ferraz, K. Hsieh, R. Thazhath, P. Martin, J. Hespanha, and M. Srivastava, "D-SLATS: Distributed Simultaneous Localization and Time Synchronization," In *ACM MobiHoc 2017, the Eighteenth International Symposium on Mobile Ad Hoc Networking and Computing*. ACM, 2017.

[43] H. Ferraz, A. Alanwar, M. Srivastava, and J. P. Hespanha, "Node Localization Based on Distributed Constrained Optimization using Jacobi's Method," In *Decision and Control (CDC), 2017 IEEE 56th Annual Conference on*. IEEE, 2017.

[44] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," In *arXiv preprint arXiv:1412.6572*, 2014.

[45] M. Pajic, J. Weimer, N. Bezzo, P. Tabuada, O. Sokolsky, I. Lee, and G. J. Pappas, "Robustness of attack-resilient state estimators," In *2014 ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, pp. 163–174, April 2014.

[46] H. Fawzi, P. Tabuada, and S. Diggavi, "Secure estimation and control for cyber-physical systems under adversarial attacks," In *IEEE Transactions on Automatic Control*, Vol. 59, No. 6, pp. 1454–1467, June 2014, arXiv: 1205.5073. [Online]. Available: http://arxiv.org/abs/1205.5073

[47] "scipy.optimize.minimize - SciPy v1.0.0 Reference Guide." [Online]. Available: https://docs.scipy.org/doc/scipy-1.0.0/reference/generated/scipy.optimize.minimize.html

[48] "What is Docker?" April 2018. [Online]. Available: https://www.docker.com/what-docker

[49] "OpenCV library." [Online]. Available: https://opencv.org/

[50] "TensorFlow." [Online]. Available: https://www.tensorflow.org/

[51] "2018 CPS Challenge | CPS-VO." [Online]. Available: https://cps-vo.org/group/CPSchallenge